NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A PROGRAM MANAGER'S GUIDE FOR SOFTWARE COST ESTIMATING

by

Andrew L. Dobbs

December 2002

Thesis Advisor:	Brad Naegle
Second Reader:	Latika Becker

Approved for public release, distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: A Program Manager's Guide for Software Cost Estimating			**5. FUNDING NUMBERS**
6. AUTHOR(S)			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release, distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT ***(maximum 200 words)***

This thesis will assist current and future program managers by outlining a process to ensure the software cost estimates developed for a system will be credible and supportable throughout the life of the program. This thesis also identifies many of the problems associated with software cost estimating and recommends potential solutions.

One of the critical parameters for estimating software cost is the quantity of source lines of code (SLOC) required in the program. Therefore, this thesis examines the software cost implications of improperly estimating SLOC and function points. Some of the other parameters required to estimate the software cost include language, functionality, application, software processes maturity, programmer skill level, design and reuse, productivity factors, complexity, utilization and schedules. Many of these parameters overlap. For example, both the complexity of the code and skill level of the programmer directly impacts the productivity and schedule of the program.

This thesis provides a broad view of the software cost estimating process. In the reference and appendix section, a list of valuable resources including commercial estimating models is provided for further assistance.

14. SUBJECT TERMS Software Cost Estimating, Software Development, Metrics, Source Lines of Code,			**15. NUMBER OF PAGES** 79
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A PROGRAM MANAGER'S GUIDE FOR SOFTWARE COST ESTIMATING

Andrew L. Dobbs
GS-13, Department of the Army
B.S., Athens State College, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN PROGRAM MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
December 2002

Author: Andrew L. Dobbs

Approved by: Brad Naegle
Principle Advisor

Latika Becker, Ph.D.
Associate Advisor

Douglas A. Brook, Ph.D.
Dean, Graduate School of Business & Public Policy

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis will assist current and future program managers by outlining a process to ensure the software cost estimates developed for a system will be credible and supportable throughout the life of the program. This thesis also identifies many of the problems associated with software cost estimating and recommends potential solutions.

One of the critical parameters for estimating software cost is the quantity of source lines of code (SLOC) required in the program. Therefore, this thesis examines the software cost implications of improperly estimating SLOC and function points. Some of the other parameters required to estimate the software cost include language, functionality, application, software processes maturity, programmer skill level, design and reuse, productivity factors, complexity, utilization and schedules. Many of these parameters overlap. For example, both the complexity of the code and skill level of the programmer directly impacts the productivity and schedule of the program.

This thesis provides a broad view of the software cost estimating process. In the reference and appendix section, a list of valuable resources, including commercial estimating models, is provided for further assistance.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACRONYMS

ACAT	Acquisition Category
CASE	Computer Aided Software Engineering
CAIG	Cost Analysis Improvement Group
CARD	Cost Analysis Requirements Document
CER	Cost Estimating Relationship
CMM	Capability Maturity Model
COCOMO	Constructive Cost Model
CSCI	Computer Software Configuration Item
DoD	Department of Defense
EIA	Electronics Engineers Association
GUI	Graphical User Interface
HOL	Higher Order Language
ICE	Independent Cost Estimate
IEEE/EIA	Institute of Electrical
IPT	Integrated Product Team
ISO	International Organization for Standardization
LCCE	Life Cycle Cost Estimate
MAIS	Major Automated Information System
MDAP	Major Defense Acquisition Program
ORD	Operational Requirements Document
OSD	Office of the Secretary of Defense
PC	Personal Computer
REVIC	Revised Intermediate Constructive Cost Model
RFP	Request for Proposal
SA	Software Acquisition
SEER	System Evaluation and Estimation Resource
SEI	Software Engineering Institute
SETA	Systems Engineering Technical Assistance
SLIM	Software Life Cycle Management

SLOC	Source Lines of Code
SU	Software Unit

ACKNOWLEDGMENTS

I would like to thank Professor Brad Naegle for his guidance and support during the preparation of this thesis. His understanding of the software development process was immensely helpful.

I would like to thank Dr. Latika Becker for agreeing to be my associate thesis advisor. Her time, patience and professionalism were greatly appreciated.

I would like to thank Jack Calvert for being my mentor for the past 11 years. His storehouse of data and expertise was helpful in preparing this thesis.

I would like to thank my supervisor, Richard Brown, for giving me the opportunity to advance my career through education.

I would like to thank Gary and Beverly Fuller for their assistance and guidance.

I would also like to thank the following individuals for taking the time to provide me with information.

Professor Dave Matthews
Jayson Wilson
Ken Shipman
Martha Spurlock
Randy Mills
Robbie Holcomb
Ed Strange
Jerome Olerich

Lastly, I would like to thank my wife, Debbie Dobbs, for being supportive of me for the past two years. Her help in proofreading and editing this thesis was a tremendous help.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Historically, software costs estimates have consistently been underestimated. The purpose of this thesis was to identify why these estimates were inaccurate. The methods used to collect data involved interviews with cost analysts, program managers, and educators. Then two data sources were used to validate the problems identified by the professionals. Then, books, trade journals, briefs to industry, software manuals, and Internet sources were used as part of the analysis. The thesis is a tool that can be used to assist current and future program managers by outlining processes to ensure the software cost estimates developed for a system will be credible and supportable throughout the life of the program.

One of the critical parameters for estimating software cost is the quantity of source lines of code (SLOC) required in the program. Therefore, this thesis examines the how the various phases of the program can impact the accuracy of the SLOC estimates. Some of the other parameters required to estimate the software cost include language, functionality, application, software processes maturity, programmer skill level, design and reuse, productivity factors, complexity, utilization and schedules. Many of these parameters overlap. For example, both the complexity of the code and skill level of the programmer directly impacts the productivity and schedule of the program.

This thesis identifies processes and plans that will improve the overall software development and result in more accurate estimates. Chapter V includes valuable resources for the program managers, and Appendix B includes a list of some of the more current software cost estimating models.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to educate program managers on software cost estimating. Software cost can comprise as much as 90 percent of some programs. [Ref.22] Therefore, understanding the software cost estimating process, and what drives the cost, is vital for the program manager to successfully manage the program. Because most program managers receive only a few weeks of formal education on the software acquisition process, few fully understand the magnitude of developing a cost estimate. The DoD schools recognize this problem and are currently adding and reviewing courses that will improve the education of software acquisition managers. [Ref. 1]

The accuracy of the software cost estimate is directly related to how well the program's software development is managed. Unfortunately, there is an alarming rate of software development failures, as described in the next few paragraphs. In 1995, the Standish Group, a firm that routinely conducts market and technology research for Fortune 500 companies, government agencies and major universities, conducted a study on the success and failure rates of software projects. The study sample size was 365 respondents and represented 8,380 software applications. Of these, only 16.2 percent of the projects were completed on-time and on-budget; 52.7 percent were completed but were over-budget, over the time estimate, and did not have full functionality; and the remaining 31.1 percent were cancelled during the development cycle. [Ref. 2]

Another finding in the Standish study reported that 52.7 percent of the projects cost 189 percent more than their original estimate. In addition to cost overruns, one third of the projects also experienced time overruns by 200-300 percent. The primary reason for these cost and schedule problems is that for every 100 projects, there were 94 restarts. This thesis examines why there are so many restarts, and what if anything can be done to reduce this trend.

Program uncertainties decrease as the development matures and advances through the various phases, and as these uncertainties are decreased, ultimately cost estimates will improve Figure 1. [Ref. 3] Many of the uncertainties can translate into project restarts.

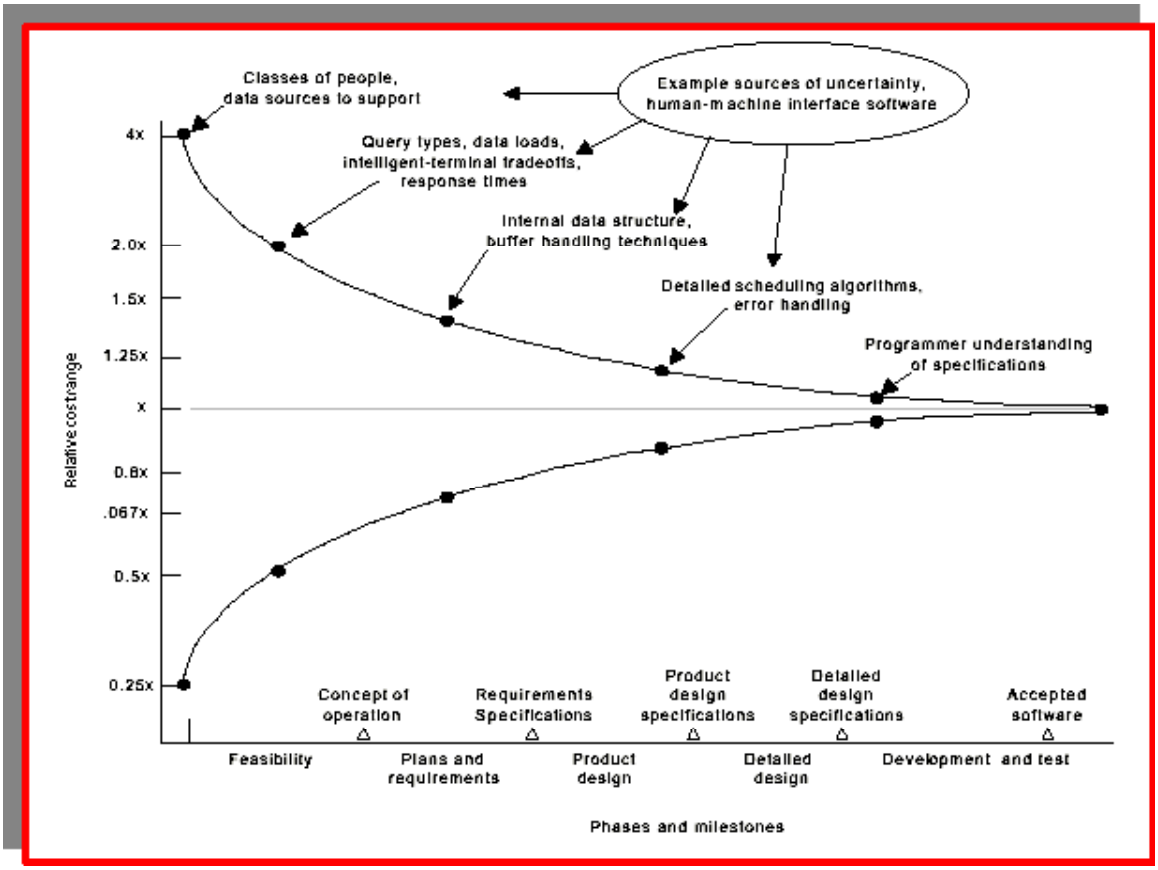


Figure 1. Software Cost Estimation Accuracy Versus Phase [From: Ref. 3]

As a program manager, you may hear that some commercial models are estimating software cost within 75-80 percent of the actual costs. Compared to the Standish Group results, 75-80 percent of actual cost is very good. However, the successes of these models are dependent upon how well the program is defined, and the accuracy of the initial SLOC estimates.

The models are also improved through a process called calibration. Calibration of the model involves inputting historical cost data of similar programs that the model will use as a basis to estimate future cost. However, if the database does not include similar projects, then the probability of the estimate falling within the 75-80 percent accuracy range is unlikely.

This thesis outlines processes that will improve the probability of success for the software intensive acquisition program. The result of following these processes should ultimately improve the software cost estimate.

B. BACKGROUND

Until recently, the Department of Defense (DoD) 5000.2-R required every Major Defense Acquisition Program (MDAP) and Major Automated Information System (MAIS) program to prepare a Life Cycle Cost Estimate (LCCE) prior to each milestone review. [Ref. 4] The Office of the Secretary of Defense (OSD) cancelled the DoD 5000 series on 30 October 2002, citing guidance that was overly prescriptive and that did not represent an acquisition policy environment encouraging efficiency, creativity, and innovation. [Ref. 5]

However, in order to provide guidance to the MDAPs and MAISs, the OSD immediately released the *Interim Defense Acquisition Guidebook.* For all practical purposes, the new guidebook is the same as the DoD 5000.2-R, and once again, requires MDAPs and MAISs to continue preparing a LCCE. [Ref. 6] Whether it is required or not, a program needs to have an estimate to track the progress of the project. The estimate will also be useful in defending and justifying the continuation the program

The LCCE is a comprehensive cost estimate that includes all costs associated with the program for its complete life cycle including both contractor and Government in-house costs for program management support. The LCCE also includes, development, test, training, deployment, operational and maintenance cost. Software related costs are included throughout the life cycle process. [Ref. 6]

C. SCOPE

This thesis identifies and analyzes significant software development issues facing program managers, and recommends potential solutions. This thesis also will examine processes that will improve the initial design requirements that are required to effectively estimate software cost.

D. RESEARCH QUESTIONS

1. Primary Research Question

What are the problems associated with software cost estimating and what solutions are available to the program manager?

2. Secondary Research Questions

What are the primary metrics for estimating software costs?

How does software programming productivity impact schedule and cost?

What are software reuse considerations?

How do you estimate software maintenance and support cost?

What are the recommended models for estimating software costs?

E. METHODOLOGY

Thesis research involved telephone and face-to-face interviews with program managers, software engineers, instructors, and cost analysts who had experience with software acquisition. These individuals provided valuable insight into the software acquisition process. Other material reviewed for this thesis includes software engineering textbooks, professional journals, software cost model manuals, symposium briefings and Internet-based software web sites.

F. ORGANIZATION

Chapter II familiarizes the reader with methodologies, models and process that are required to prepare software cost estimates.

Chapter III presents the data, outlines the primary causes of inaccurate software cost estimates and lays the foundation for analysis. Chapter IV analyzes the data and recommends potential solutions to the estimating problems. Chapter V summarizes the analysis and provides recommendations to improve the overall software cost estimating process.

G. BENEFITS OF THE ANALYSIS

The analysis provides the program managers with insight into software cost estimating that will result in more accurate cost estimates. By improving the program's cost estimate, resources can be allocated to ensure the software project remains on schedule, within budget, and delivered with the desired capabilities.

II. METHODOLOGIES, MODELS AND PROCESSESS

The next sections will provide background information helpful in software cost estimating. It includes methodologies to develop software requirements, cost estimating methods and processes, and definitions to key software terms.

A. PRIMARY SOFTWARE DEVELOPMENT METHODOLGIES

Software can be written in many languages. For example, Machine Language is code written in 0's and 1's; Assembly Language is written in English and assembled into Machine Language; Higher-Order Language (HOL) is similar to Assembly, but usually easier to read and write (i.e., FORTRAN, COBOL, Basic, Ada, C++, and Java), and Very High-Level languages, also called 4th Generation Language, are written to resemble the spoken language that includes programs for spreadsheets, word processors and graphical user interfaces (GUI). [Ref. 7]

On April 29 1997, DOD lifted a 1987 policy that required all military systems be developed in Ada. The National Research Council reported that Ada (version 1995) was superior over C, C++, and Java when applications required real-time processing, high-assurance, and high-reliability for weapon systems. However, the report also noted that Ada came in second behind COBOL for administration applications. [Ref. 8] Examples of the various languages and a process required to go from the spoken language down to the binary code required to execute the program is shown in Figure 2.

Regardless of the language used in the development, all software programs must be developed through a systematic approach. One recommended approach is the Institute of Electrical and Electronics Engineers/Electronic Industries Association's (IEEE/EIA) 12207, "Standard for Information Technology--Software Life Cycle Processes" or International Organization for Standardization (ISO) 12207. This approach is a generic software process that recommends a set of development activities and documentation alternatives for software intensive programs. [Ref. 7]

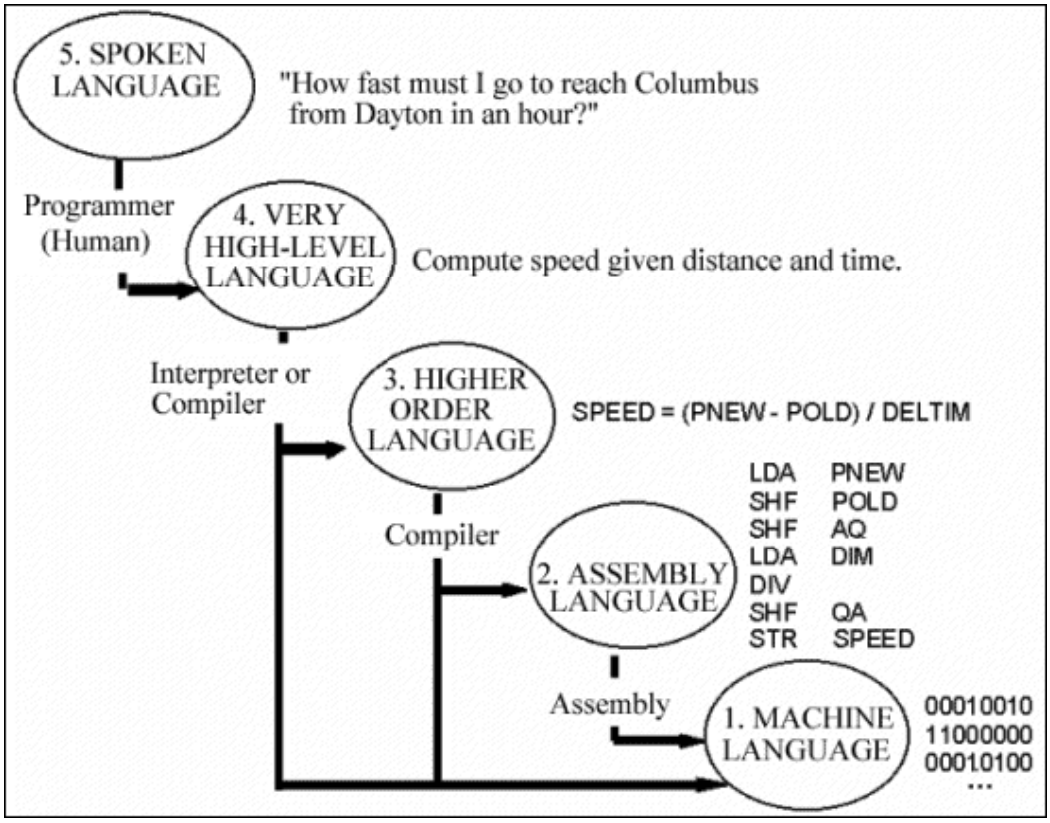


Figure 2. Software Language Translation Process [From: Ref. 7]

The ISO 12207 Primary life cycle process begins with: 1) acquisition process; 2) supply process (providing software to the customer that meets the agreed requirements); 3) development process (includes system and software requirements analysis, architectural and detailed software design, software coding and testing, software integration, qualification testing, installation and acceptance); 4) operation process; and 5) maintenance process. This process is compatible with most of the leading software development methodologies outlined below. The ISO 12207 primarily serves as a checklist to ensure that all aspects of the software development are considered. The methodology selected will have a significant impact on the development and maintenance cost. [Ref. 7]

1. Waterfall/Traditional

The Waterfall methodology was developed in 1970 by W.W. Royce, and considered to be the first formal disciplined approach for software development. [Ref. 3] This methodology assumes that all the requirements are known up front and therefore a

complete design of the program can be achieved and the process of coding the software can begin. Unfortunately, this methodology does not work well with the majority of advanced technology programs. [Ref. 7]

For example, using this methodology may delay the delivery of a missile system until all of the weapon capabilities are achieved. If a different development approach is used, the missile could be delivered with limited capabilities at an earlier date. For instance, the missile could be delivered immediately with the limited capability to destroy fixed wing aircraft, even though the missile still lacks the capability to destroy helicopters. The Army's Patriot missile system has been adding capabilities like this example over the years using incremental software builds with tremendous success.

Another problem with the Waterfall methodology is that many of the errors in the software will not be discovered until the end of the development. At this point, correcting these errors will be time-consuming and costly. The Waterfall software development process is shown in Figure 3. [Ref. 9]

2. Evolutionary Development

Evolutionary development begins the design process with only the core capabilities and delivers an initial operational product. The next step in the process is to add more functionality and refine the previous design. This process continues until the program is complete. The advantages of this process are that it places a working product in the hands of the user and allows them to provide input into future designs. The disadvantage of the Evolutionary method is that it usually takes more time to complete the project. The Evolutionary Development process with user involvement is shown in Figure 4. [Ref. 7]

3. Incremental Development

Software is developed in a series of increments of increasing functional capability. Like the Evolutionary methodology, the Incremental Development methodology lets the user get involved early through a build-and-test process. The Incremental approach is best suited when user requirements can be fully defined, or when

factors such as technical risks, funding instability, schedule uncertainties, or program size warrant a phased approach. Other advantages to this methodology are a reduction in risk and a firm foundation to meet the requirements of the remaining software builds/releases. The primary disadvantage for this methodology is that it is not always easy to break up a design into useful increments. [Ref. 7] The Incremental Development process of build and release until complete is shown in Figure 5.

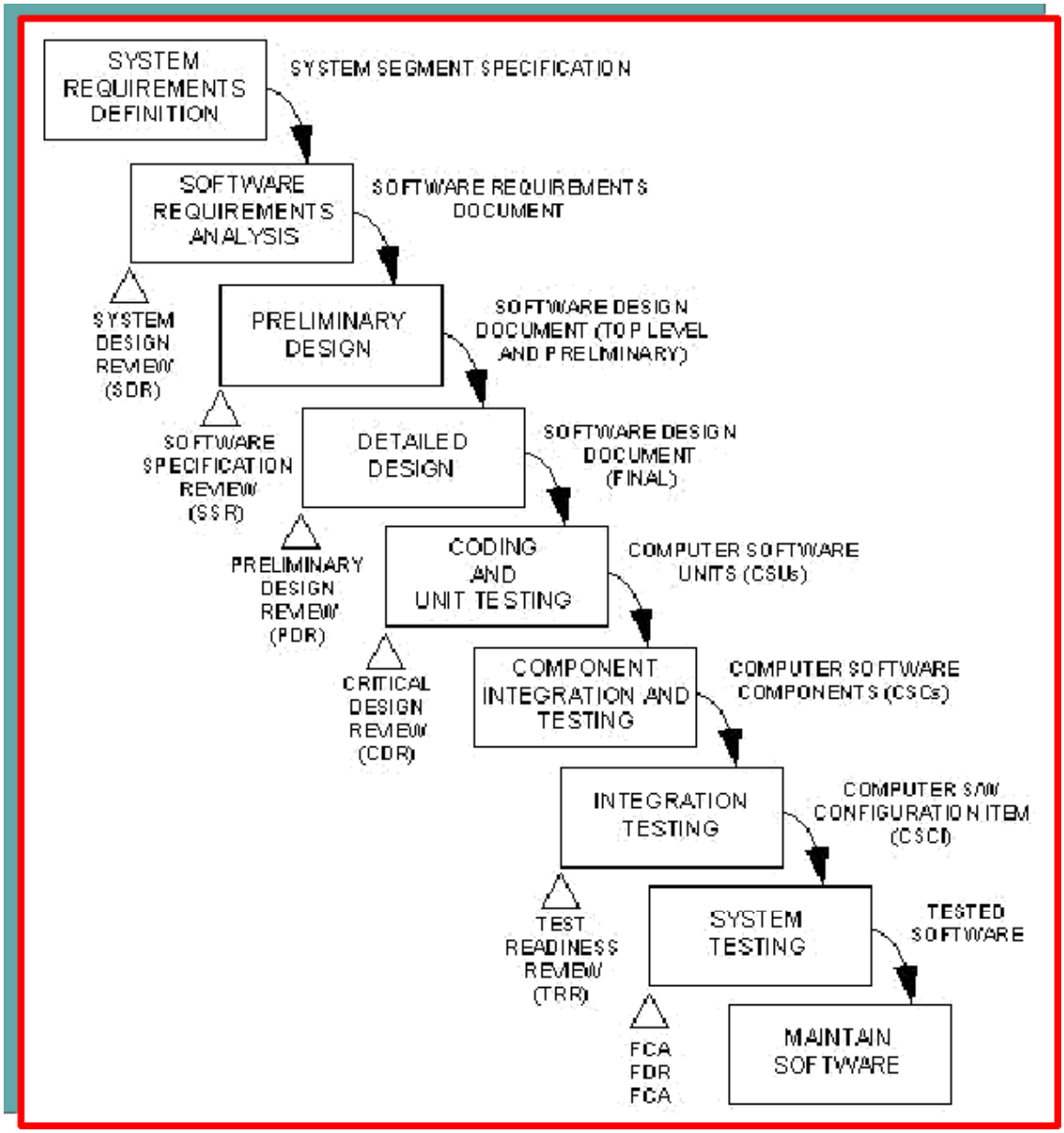


Figure 3. Software Waterfall Model Development [From: Ref. 9]

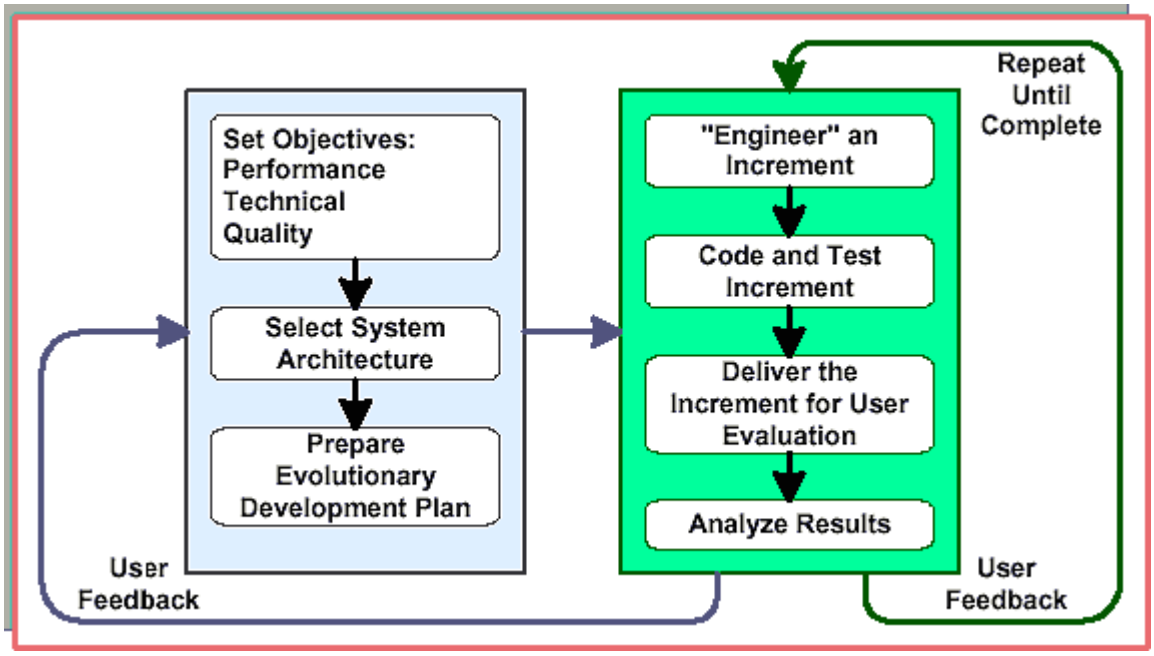


Figure 4. Evolutionary Model With User Involvement [From: Ref. 9]

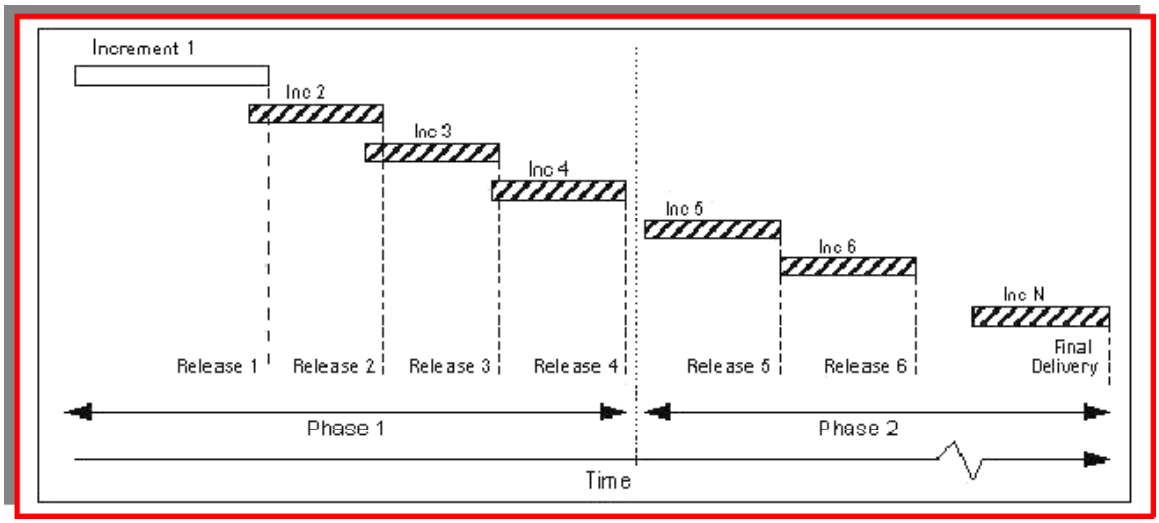


Figure 5. Incremental Software Development [From: Ref. 9]

4. Prototyping Development

The Prototyping Development methodology is similar to a hardware bread-board design where basic technical components are integrated to establish that the pieces work together. Like bread-boarding, prototyping software developments are relatively "low fidelity" compared to the eventual system. This procedure provides the user with an experimental system to evaluate their initial requirements. Once the initial requirements are understood, the final requirements can be easily determined. Computer-aided

software engineering (CASE) tools are extremely beneficial in developing prototype systems.

CASE tools help the contractor efficiently develop relatively defect free, easily modified, quality software. Besides providing the user with a prototype system, CASE tools can also be used for planning and estimation, requirements analysis and design, architectural design flexibility, improving productivity, shortening lead time, and freeing up software developers from mundane tasks.

5. Spiral Development

The Spiral Development model was developed by Dr. Barry Boehm in 1987 as a risk-reduction approach to software development. This methodology illustrates the software development as a spiral with radial distance as a measure of cost or effort, and angular displacement as a measure of progress. Looking at Figure 6., starting at the center with project definition and working clockwise through the spiral, each cycle includes a review of objectives, alternatives, constraints, various analyses (including risk analysis), and one or more products are delivered. [Ref. 9]

The advantage to the Spiral model is that it emphasizes evaluation of alternatives using risk analysis, and provides flexibility to the software development process. This is accomplished by using basic Waterfall building blocks and Evolutionary/Incremental prototyping approaches to complete the software development. [Ref. 7]

6. Object-Oriented Development

With the Object-Oriented Development methodology, procedures and data are combined into unified objects. The Object-Oriented system is a collection of classes and objects and how they relate to each other. As an example of this class/object relationship, Figure 7. illustrates the "class" as missile, and a guidance system as an "object". Because the guidance system is a member or subset of the "class" missile, the guidance system will inherit all of the same attributes from the "class" missile, such as cost, dimensions, weight, range, and any other possible attributes. [Ref. 10]

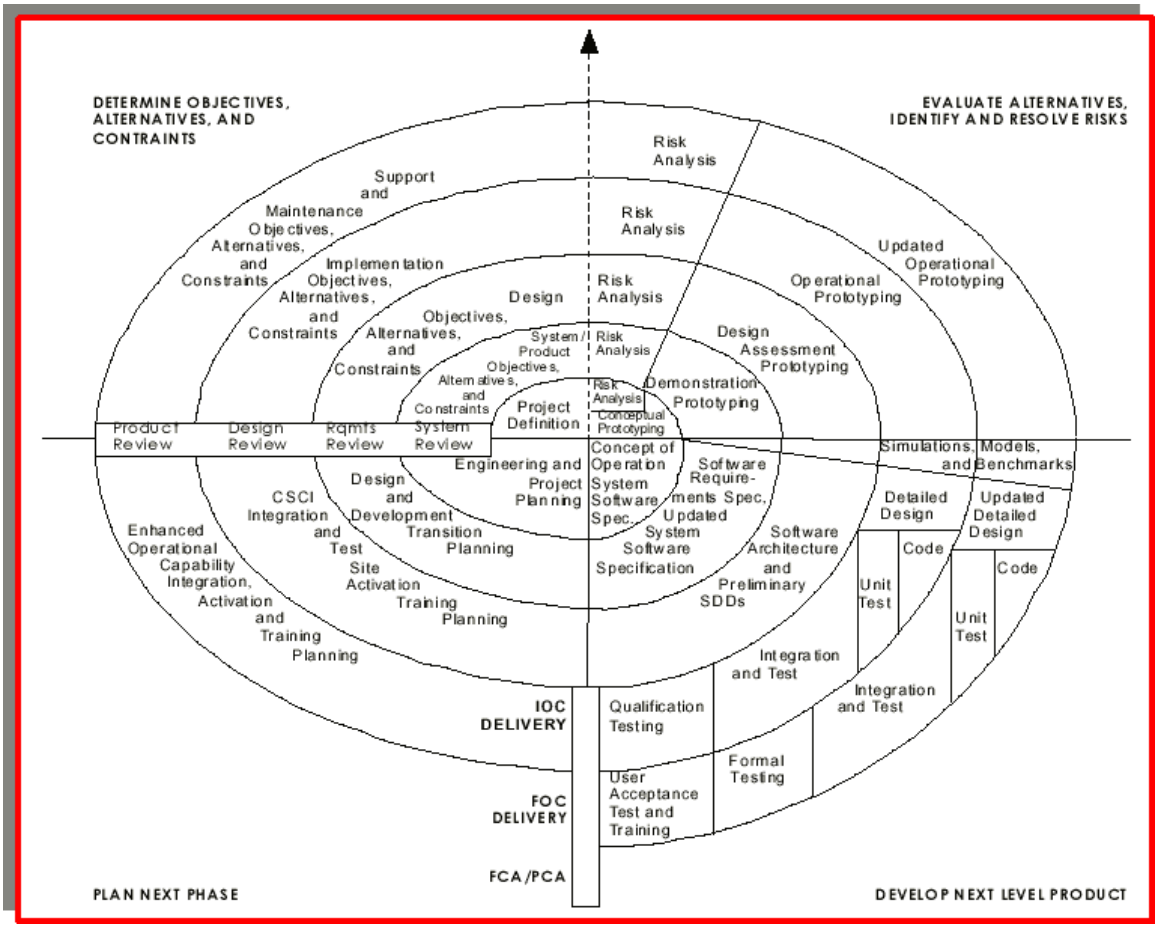


Figure 6. Spiral Development Methodology [From: Ref. 9]

The Object-Oriented programming is not usually considered a stand-alone development process. An Evolutionary/Spiral type methodology should be used with the Object-Oriented process, because it would be difficult to define all the required classes for a major system or product in a single iteration. [Ref. 10]

For example, beginning in the center of the Spiral model, communication with the customer helps define the program, and identify the classes or major design points. Then, planning and risk analysis establish the foundation for the Object-Oriented project plan. All technical work that follows will be accomplished through an iterative approach. Object-Oriented programming always searches a library of classes to determine if reuse software is available. If not, the Object-Oriented method begins the process of analysis, design, programming, and testing to create the new class, and all the objects derived from that class. The new class is added to the library, and the process continues until the end of the development. [Ref. 10]

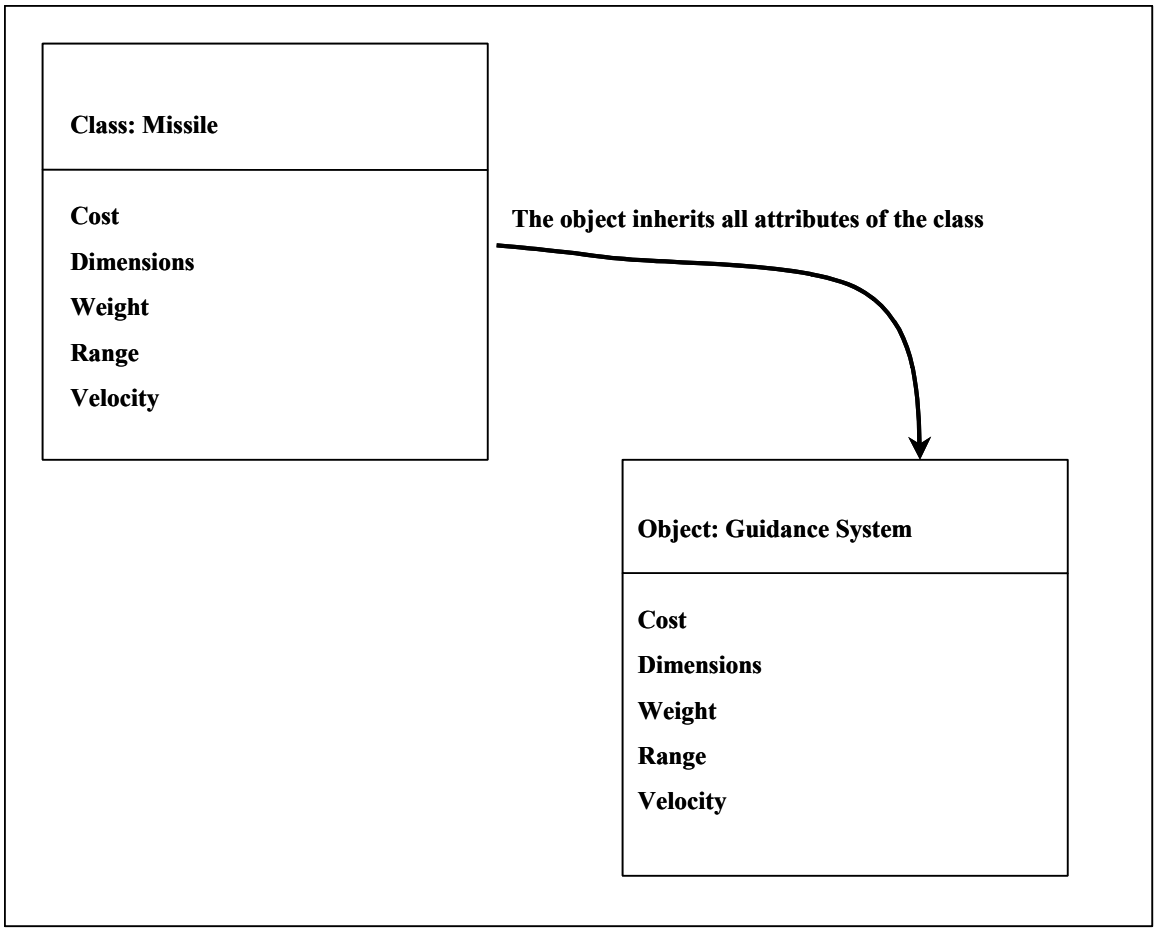


Figure 7. Object-Oriented Inheritance From Class To Object [From: Ref.10]

B. PRIMARY METHODS TO ESTIMATE SOFTWARE COSTS

1. Analogy

With an Analogy method, lines of code and cost estimates are based on a historical database of similar type programs. This methodology is usually the most accurate means during the beginning of a program. For example, to estimate the lines of code and software cost associated with a new cruise missile interceptor, the analyst would search the database for an existing missiles with similar launch, flight, fusing and warhead characteristics. If one or more similar missiles are included in the database, the software development and maintenance cost could then be used as a basis for the new estimate. However, due to the advanced technologies within defense programs, most databases do not include similar projects. The Analogy technique is often used as a secondary method to check other estimates for reasonability. [Ref. 7]

2. Parametric Estimating

Parametric estimating involves using mathematical equations based on cost estimating relationships (CERs) to estimate software costs. As a top-level example of a CER, an analyst may provide a program manager a quick estimate based on the SLOC anticipated for development of a missile multiplied by a cost figure (i.e., 14,000 SLOC multiplied by $200 yields an estimated cost of $2.8 million). The CERs express cost as a function of one or more cost driving variables, and are developed from historical databases of similar software projects.

In order to develop a CER, the analyst begins by analyzing a project to determine what factors could influence the cost of the project. Using the missile analogy again, the analyst may speculate that SLOC developed and maintained for a missile could potentially be used to estimate the total software development cost of a new missile. At this point, the analyst would collect data to validate those assumptions.

Care must be taken to ensure that the data is normalized, for instance comparing the cost per SLOC of a missile developed and built five years ago will be different from one built last year. Therefore, the cost of the missile built five years ago will be escalated to a more current year. Once this is accomplished, the relationship can be tested, by plotting the normalized historical data from all of the different completed missiles. If the resulting graph is linear, then there is a good chance the relationship is valid.

A larger database will statistically provide a higher confidence that the relationship is valid. There are user-friendly statistical software programs available to aid in this process. Figure CER shows the process required to create a CER. The Parametric estimating method is normally used to estimate the overall system or at the computer software configuration items (CSCI) level. [Ref. 7]

A CSCI is defined as a collection of software that satisfies a common end use function. Typically, when the size of the overall system or CSCI exceeds 100,000 lines of code, it is further partitioned into more manageable tiers called software units (SUs). Parametric software cost estimating models are fairly easy to use and can provide quick estimates that in most cases are more accurate than other methodologies. [Ref. 7]

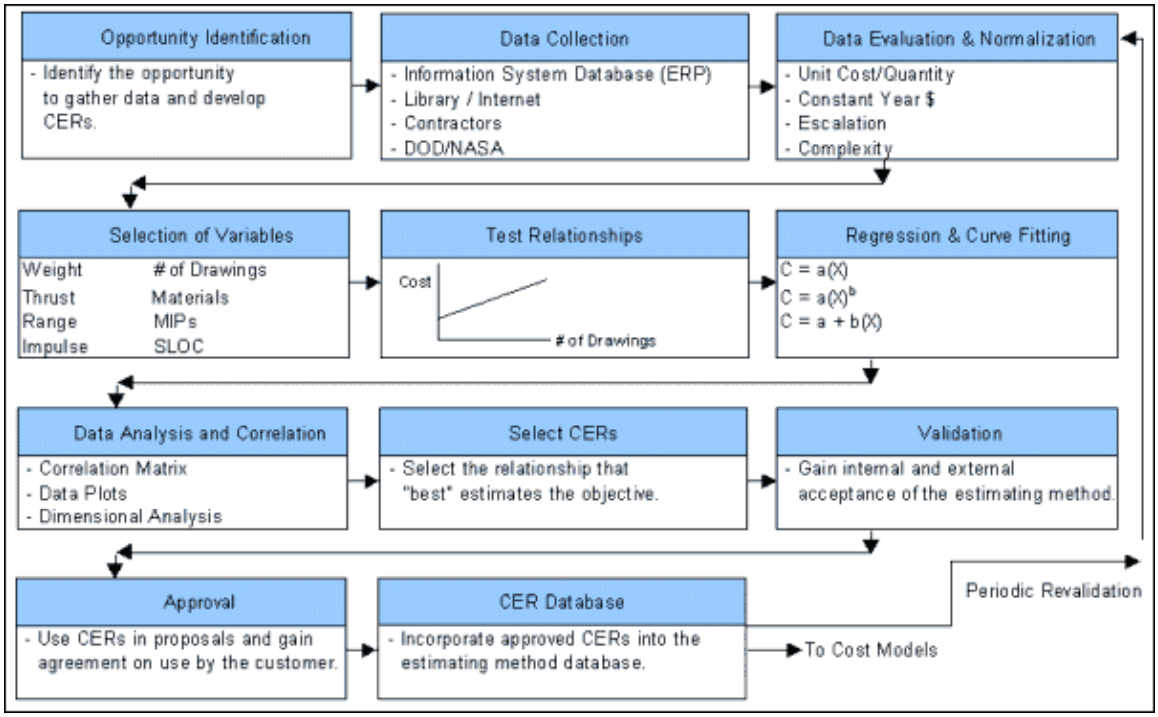


Figure 8. Cost Estimating Relationship Development Process [From: Ref. 7]

A detailed process to normalize data required to create a CER is shown in Figure 9. There are also polices and procedures for calibrating and validating the software cost estimating models. At the time this thesis was being written, OSD was re-writing/re-placing the 5000 series that included most of the policies and requirements required for Acquisition Category (ACAT) programs.

3. Bottoms-up Approach

The Bottoms-Up estimating approach requires the project to be sufficiently designed to permit reasonable estimates at the SU level. These detailed SU cost estimates are then added up to the CSCI level and ultimately at the project level. This approach is time consuming and only as good as the design. As the quality and design of the project improve, so does the estimate. This method does make it easier to track the success of the project, because of the level of detail. [Ref. 7]

One of the major disadvantages of this method is that many times the costs for software integration activities are not captured. Just like the other methodologies, historical data is not always available to compare projects. The detailed process required to complete a Bottoms Up estimate is illustrated in Figure 10.

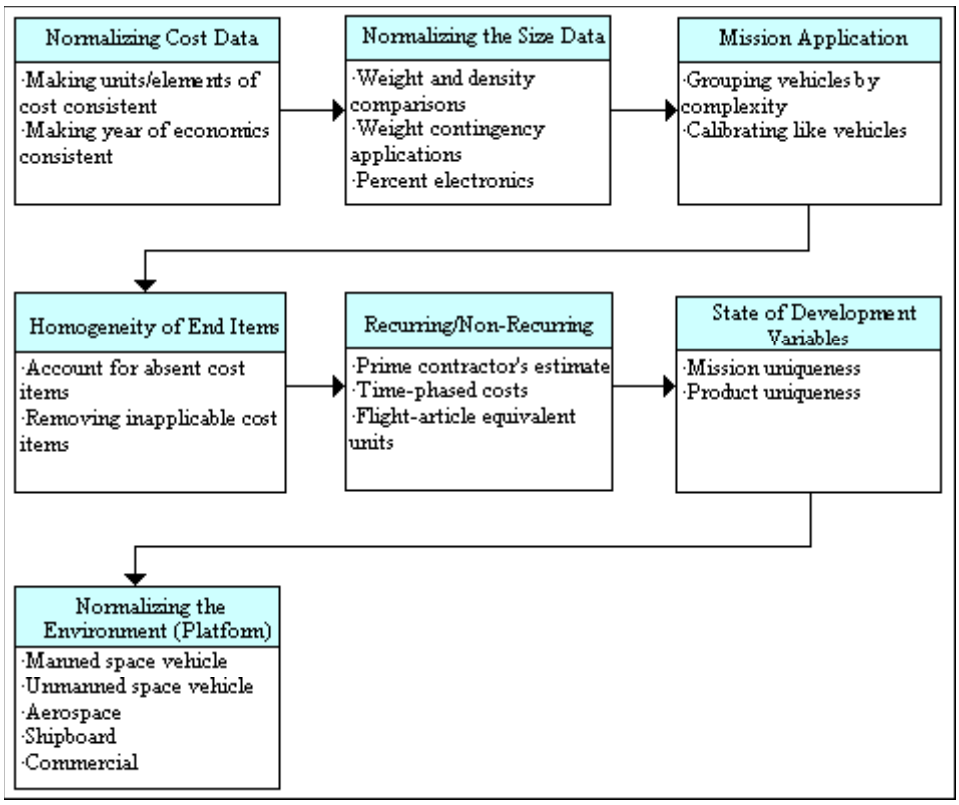


Figure 9. Normalization of Data [From: Ref. 7]

4. Engineering Judgment

The Engineering Judgment estimate is basically what is considered an educated guess. With this approach, experienced software engineers estimate the size of the code based on previous software experience and their knowledge of functions to be developed. Several cost analysts with over twenty years experience in the estimating profession, have stated that many times, because of poor program definition, there has been no real basis for the estimate.

Therefore, an engineering estimate, or what is fondly referred to a "back of the envelope" estimate, is as good as any other estimating methodology. For example, the analyst might multiply a current cost do develop code by the estimated SLOC count, and then double or triple the estimate. Unfortunately, this is the methodology with which most software estimates are forced to begin. This method is useful for determining inputs into other models, but not sufficient for use as the basis of an estimate. [Ref. 7]

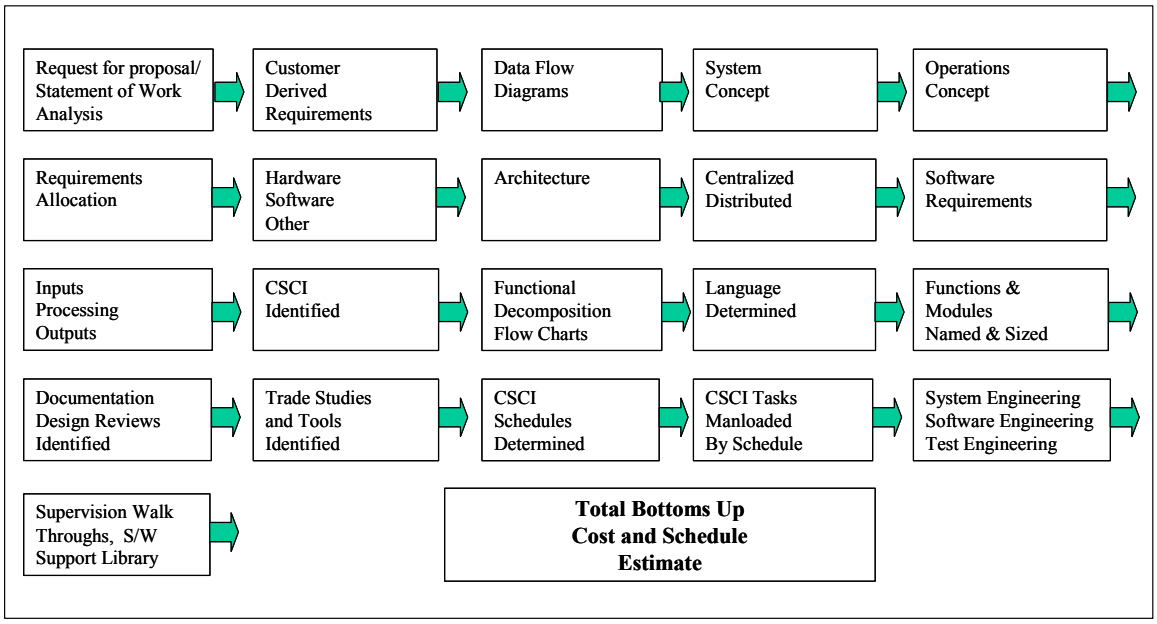


Figure 10. "Bottoms Up" Software Estimating Process [From: Ref.11]

C. SOFTWARE COST ESTIMATE PROCESS

Prior to beginning the process of preparing the cost estimate, the program office is required to prepare a Cost Analysis Requirements Document (CARD). This document provides a description of the most important features down to the Work Breakdown Structure (WBS) level of the program and serves as the basis for all cost estimates. It also defines and provides quantitative descriptions of the program characteristics. [Ref. 4,18] Because of the magnitude of this document, an Integrated Product Team (IPT) should be established to prepare the CARD. Other documents such as the Operational Requirements Document (ORD) and Performance Specification documents will assist in the development of the CARD. The IPT should include representatives from all of the critical areas of the program.

The Office Secretary Of Defense, Cost Analysis Improvement Group (CAIG,) requires that a draft of this document be delivered to them 180 days prior to a milestone review and a final copy 45 days prior to the review. The CAIG will use the CARD to prepare an Independent Cost Estimate (ICE) of the program. [Ref. 4]

The CARD is a living document, and it is critical that configuration control be maintained throughout the life of the program. The program office is required to prepare

a new CARD if there are any major changes to the exiting program or if alternative designs are evaluated. For example, the initial CARD was based on a missile that only intercepts fixed-wing aircraft. Now the User has delivered a new ORD that requires the missile to also intercept helicopters, unmanned vehicles, and cruise missiles. Therefore, a new CARD must be developed to incorporate the additional requirements. This is a prime example of "requirements creep," and in most cases a "new start" is required.

The software cost estimating process in Table 1. provides a systematic approach to successfully prepare the project software cost estimate. This process should be managed through the IPT. The process begins with the IPT breaking the total software development project into manageable lower-level CSCI and SU elements. Then, the team can determine the scope (size) of each element, assess the software development environments, and perform assessments of alternatives and risk factors.

Once the functional decomposition is complete, the various environments can be quantified, evaluated, and "high/low" boundaries can be assessed. This will establish the initial parameters required for the baseline estimates of cost, schedule, resources, and support. This process was designed to prepare the project for a contactor bid. However, it can be easily adapted for any phase of the development. [Ref. 9]

1. Design Baseline

The most effective method of managing a large software development is to decompose the project into manageable parts. Decomposition can be accomplished by two different methods. One involves a functional decomposition of the program, which divides it into basic components from the user's perspective. The other is design-decomposition, which divides the project into software components or modules. Both of these methods make it easier for the analyst to realistically estimate size, time, and manpower required for each function. This has been referred to as the "divide and conquer" method, and is also in line with Spiral and Evolutionary development methods. [Ref. 9]

Phase	Major Activity	Specific Products
Design Baseline	Define a point of sufficient precision to identify the number of CSCIs and the required functionality of each.	List of CSCIs, functionality and similar completed projects or CSCIs.
Size Baseline	Using the products from the Design phase, define the expected size for each CSCI.	List of CSCIs with appropriate size information
Environmental Baseline	Using the products from the two previous phases, determine the environmental characteristics required and available to perform the job.	List of software cost model parameters and their initial settings along with a written rational for each.
Software Baseline Estimate	Using the size and environment products, make a software cost model run (using whatever model best satisfies the organizations needs).	Output from the software cost model showing schedule an cost information.
Project Baseline	Using the output from the Software Baseline Estimate phase, add those elements not included in the particular software cost model (each model has a specific set of items not included in the estimate) and remove elements included in the estimate that are not part of the project.	A complete estimate of the costs and schedule for the software portion of the project.
Risk Analysis	Determine the cost/schedule risk associated with the Project Estimate. Make changes to the size or environment products to perform what-if analyses. Determine the size and/or environment setting required to validate the final software bid.	Risk assessment, risk graphs risk memorandum with Parameter-by-parameter risk explanations.
Project Bid	Perform analysis of the Project Estimate, considering such factors as expected competition, type of contract, budgetary or personnel constraints, risk analysis, etc. Convert labor and other direct charge (ODC) estimates into contractor's price and determine the Project Bid.	Project Bid.
Dynamic Cost Projection	Using existing known environment and size information, produce a revised Project Estimate and determine the remaining costs and schedule to complete for the on-going project	Cost-to-Complete, Schedule-to-Complete, Size-to-Cost.

Table 1. Software Estimating Process Elements [After: Ref. 9].

2. Software Size

Predicting the size and complexity of the software required for a program is the leading cause of cost overruns and schedule slips. [Ref. 9] Most of the cost models

reviewed for this thesis require an input for how many SLOC would be in the project. Thus, the first basic step in developing a cost estimate is determining the lines of code required for the program. The size and complexity of the program will significantly influence the resources required to estimate the program. To make this even more complicated, requirements creep will continually increase the estimated lines of code.

SLOC can have different definitions based on the user and the developer. The Software Engineering Institute recommended that DoD organization use SLOC as the first measure of software size. The SEI also recommended that program offices should establish a clear definition of what is considered a SLOC. Table 2. is provided to assist the program manager in determining which type of statements should be included and excluded from the SLOC count. For example, if the checklist in Table 2. is provided in the request for proposals, the resulting bids can be evaluated with less confusion. [Ref. 13]

Another factor to consider when calculating SLOC counts is how much of the software development will be reuse code. Reuse code is pre-existing code from another software program and used in the new development. Sometimes, incorporating reuse code into the program can save significant resources. However, if the pre-existing code is poorly designed and documented, the cost can actually be greater. The quality of the reuse code must be considered when estimating the cost. Another factor to consider is whether the code was initially designed with the intention of being reused. Usually code that was designed with the intent that it would be modular and reusable is better documented and easier to modify.

One of the earliest best-known software cost models was the Constructive Cost Model (COCOMO), released in 1981. This version discounted modified reuse code by 90 percent of what it would have cost had it been built from scratch. The new version of COCOMO released in 2000, now only discounts reuse by 50 percent. This was changed because the data and actual experience over the last few years has changed. [Ref. 12] The savings may vary based upon integration and testing required. Most estimating models are adjusted based on the application of the reuse code, the company's prior

experience, or are adjusted within the SLOC count. If the program contains a lot of reuse code, the model selected should be evaluated to determine how the code is estimated.

	Statement Type	Order of Precedence	Includes	Excludes
	When a line or statement contains more than one type, classify it as the type with the highest precedence.			
1	Executable	1	X	
2	Nonexecutable			
3	Declarations	2	X	
4	Compiler Directives	3	X	
5	Comments			
6	On their on lines	4		X
7	On lines with source code	5		X
8	Banners and nonblank spacers	6		X
9	Blank (empty) comments	7		X
10	Blank Lines	8		X
11				
12				
	How Produced		Includes	Excludes
1	Programmed		X	
2	Generated with source code generators		X	
3	Converted with automated translators		X	
4	Copied or reused without change		X	
5	Modified		X	
6	Removed			X
7				
8				

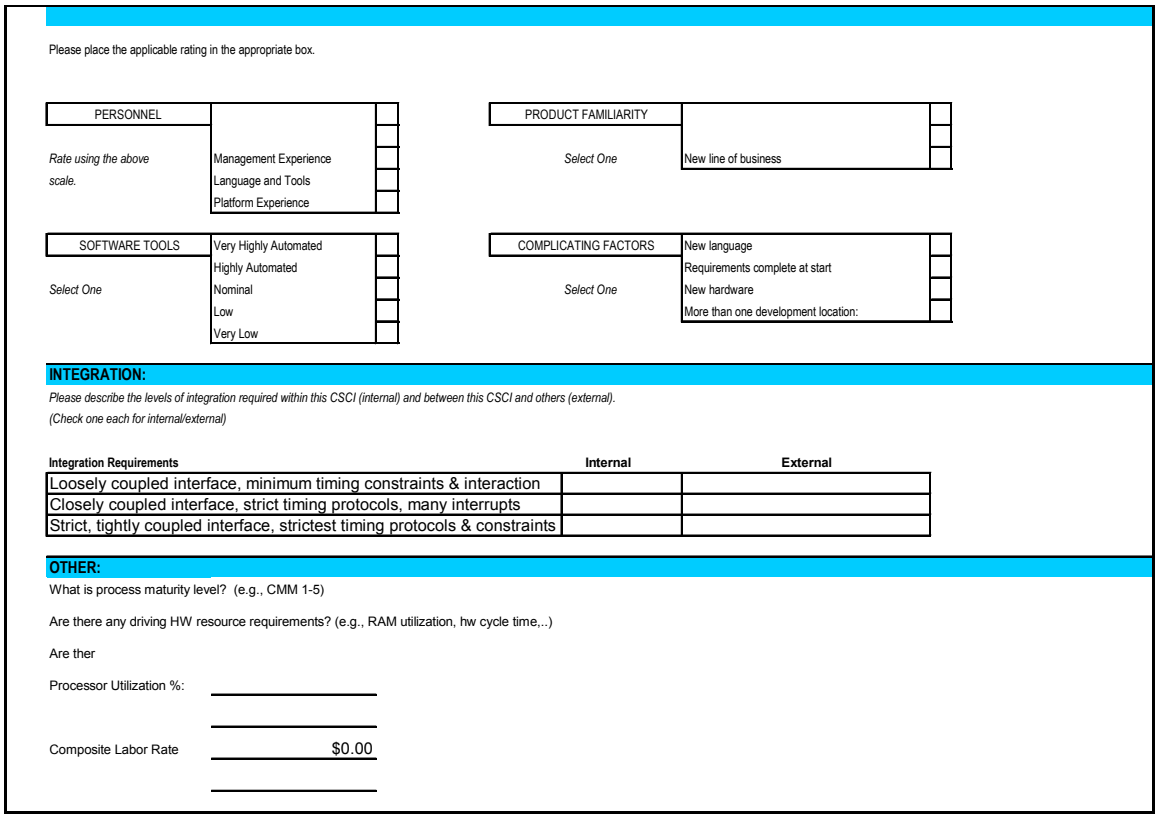
Table 2. Definition Checklist for SLOC [After: Ref. 9]

3. Environmental Inputs

There are many environmental factors that impact the overall software cost of a program. These factors are incorporated slightly different in each of the software cost models. Care should be taken to ensure that the model selected includes all of the necessary parameters that could impact program costs, and likewise, exclude any parameters that would adversely impact the program. An example of an actual questionnaire, sent out to government contractors in order to collect data to prepare a command and control software development cost estimate, is illustrated in Figure 11.

An example of a common set of environmental factors embedded in the Revised Intermediate COCOMO (REVIC) model is shown in Table 2. It is beyond the scope of

this thesis to address each one of the environmental factors, however, it will include enough information to provide an understanding of how the models adjust cost.



Please place the applicable rating in the appropriate box.

PERSONNEL		
Rate using the above scale.	Management Experience	
	Language and Tools	
	Platform Experience	

PRODUCT FAMILIARITY		
Select One	New line of business	

SOFTWARE TOOLS		
Select One	Very Highly Automated	
	Highly Automated	
	Nominal	
	Low	
	Very Low	

COMPLICATING FACTORS		
Select One	New language	
	Requirements complete at start	
	New hardware	
	More than one development location:	

INTEGRATION:

Please describe the levels of integration required within this CSCI (internal) and between this CSCI and others (external).
(Check one each for internal/external)

Integration Requirements	Internal	External
Loosely coupled interface, minimum timing constraints & interaction		
Closely coupled interface, strict timing protocols, many interrupts		
Strict, tightly coupled interface, strictest timing protocols & constraints		

OTHER:

What is process maturity level? (e.g., CMM 1-5)

Are there any driving HW resource requirements? (e.g., RAM utilization, hw cycle time,..)

Are ther

Processor Utilization %: _____

_____

Composite Labor Rate $0.00

_____

Figure 11. Example of Questionnaire to Industry [From: Ref.14]

Analyst Capability	**Required Software Reliability**
Programming Team Capability	**Database Size**
Project Application Experience	**Software Product Complexity**
Virtual Machine Experience	**Required Reusability**
Language Experience	**Modern Programming Practices**
Execution Time Constraints	**Use of Softwae Tools**
Main Storage Constraints	**Classified Security Application**
Virtual Machine Volatility	**Management Reserve for Risk**
Computer Turnaround Time	**Required Development Schedule**
Requirements Volatility	

Table 3. Types of Environmental Factors [After: Ref. 15]

The IPT approach should once again be used to evaluate these factors and assign expert-judgment values from "very-low" to "very-high," and in some cases from "very-low" to "extra extra-high," depending on the factor. These subjective factors are then

assigned numerical values to be multiplied together to effect overall software cost. The numerical values are derived from historical databases of similar type programs.

Table 3. is an example of the one of the factors from the REVIC model. The example reflects that a programmer team ranked in the 90th percentile would receive a score of "very high," because an experienced programming team can produce more code in a shorter period of time. Therefore, the database assigns the numerical multiplication factor of ".71," which would reduce the cost of the estimate. The cost analyst needs to understand exactly what each factor includes.

Rating	Skill Level	Factor
Very Low	15th Percentile	1.46
Low	35th Percentile	1.19
Nominal	55th Percentile	1.00
HI	75th Percentile	0.86
Very High	90th Percentile	0.71

Table 4. REVIC Example of Programmer Capabilities [From: Ref. 15]

The software Capability Maturity Model (CMM) is used to assess the effectiveness of a company's software processes. Table 5. outlines the different process maturity levels, describes the characteristics of each level, and shows productivity and risk projections. The DoD requires that companies must have a CMM level of three or higher to bid on contracts. This ensures but does not guarantee that contractors submitting bids have the appropriate software development teams with the experience required to deliver the product on-budget, on-schedule and fully capable. [Ref. 9]

Each of the software models account for the CMM levels in a slightly different method. For example, the new version of COCOMO II has an input for "Estimated Process Maturity Level" [COCOMO II], while PRICE S considers several factors such as efficiency, skills, familiarity and intensity of the effort. Studies have shown that companies with higher levels of CMM are more likely to perform better in the software development process. The CMM was developed based on work conducted by Watts Humphrey of the Software Engineering Institute at Carnegie Mellon University. [Ref. 7]

Maturity Level	Characteristics	Key Challenges	Results
5 *Optimized*	- Improvement fed back into the process - Automated tools used to identify weakest process elements - Numerical evidence used to apply technology to critical tasks - Rigorous defect-causal analysis and defect prevention	- Still human-intensive process - Maintain organization at optimizing level	PRODUCTIVITY QUALITY RISK
4 *Managed*	(Quantitative) - Measured process - Minimum set of quality and productivity measurements - Process data stored, analyzed, and maintained	- Changing technology - Problem analysis - Problem prevention	
3 *Defined*	(Qualitative) - Process defined and institutionalized - Software Engineering Process Group leads process improvement	- Process measurement - Process analysis - Quantitative quality plans	
2 *Repeatable*	(Intuitive) - Process dependent on individuals - Basic project controls established - Strength in doing similar work, but new challenges present major risk - Orderly framework for improvement lacking	- Training - Technical practices (reviews, testing) - Process focus (standards, process groups)	
1 *Initial*	(Ad hoc/chaotic process) - No formal procedures, cost estimates, project plans - No management mechanism to ensure procedures are followed - Tools not well integrated; change control is lax - Senior Management does not understand key issues	- Project management - Project planning - Configuration management - Software quality assurance	

Table 5. General Characteristics of Each Maturity Level of The CMM [From: Ref. 9].

For example, a CMM level three certified team with four or more years experience developing code in a given language and using modern software engineering methods would score a rating of "extra high." This "extra high" rating would automatically lower the cost for developing the software. Likewise, a development team with less than one year of experience working together may receive a rating of "low," which increases the risk and cost appropriately. [Ref. 16]

Another environmental parameter that influences several factors is the application type of the software development. Typical applications for military systems include command and control, data base management, diagnostics, graphics, message switching, mission planning, RADAR/Sensor processing, and systems engineering simulation, etc. The software cost models make adjustments to the overall cost based on application types by assigning complexity factors to product reliability, software complexity, and classification types.

For example, if the software application is being developed for a RADAR or Fire Control system, the software complexity factor would be considered "High," and the cost would be multiplied notionally by 25 percent to cover the extra cost required to develop this software. Likewise, if the software is for user interfaces or communications networks, the software complexity factor may be considered "nominal" and applications

for administrative data processing may be considered "low." There would not be an increase in cost for a nominal factor, but the "low" factor would reduce the cost notionally by 25 percent. The term "notionally" is used because each model has a slightly different method of accounting for numerical complexity factors. [Ref. 16]

.

4. Software Baseline Cost Estimate

Once the size, reuse, and environmental parameters have been determined, the analyst is now ready to estimate the program using one of the estimating methodologies identified in Chapter II. When considering which method should be used to estimate the program, the following questions should be asked: how quick does the estimate have to be completed; and how much detail is required. For example, the program should use a parametric software cost estimating model if detailed cost and schedule are required. The estimating model usually is much better than the other methods, because analysts are less likely to leave out an important parameter.

D. SOFTWARE COST ESTIMATING MODELS

There are several excellent software models available to the program office for estimating software costs. The following models are currently the most widely used in alphabetical order): COCOMO II, CostXPert, PRICE S, REVIC/SoftEST, Sage, System Evaluation and Estimation of Resources (SEER), Software Life Cycle Management (SLIM). [Ref. 19] Appendix B. includes websites where detailed information on capabilities and purchase cost can be found.

Most of the estimating models are now personal computer (PC) based, and fairly easy to operate with a little training. CostXPert, claims an estimate is possible within fifteen minutes after product installation. At least one independent evaluator said that this is entirely possible. [Ref. 20] However, most models would require some level of training to ensure maximum understanding of the tool.

When evaluating cost estimating models, the program office should determine if the model is pre-calibrated with similar applications that will yield reasonable results. It may be necessary to calibrate the model for the application. For example, the rating

scales used by COCOMO II for personnel factors, such as analyst and programmer capability, may not be suitable for a different organization. The COCOMO II developer recommends the organization have at lest five data-points to modify the multiplicative constant and ten data points for calibrating both the calibrating and the baseline exponent before changing the baseline. [Ref. 12]

When calibrating the model at the contractor level, validation of the model is accomplished by demonstrating the credibility of the parametric model prior to submitting an estimate to the government or higher tier contractor. Both calibration and validation should be conducted on a periodic basis throughout the software's development. [Ref. 7]

THIS PAGE INTENTIONALLY LEFT BLANK

III. DATA TO BE ANALYZED

The primary research question for this thesis was centered on the problems that cause software cost estimates to be inaccurate. The secondary questions were designed to determine the impact of critical cost drivers on the software cost estimate. Two different methods were used to collect data for this thesis. The first method involved interviewing professionals in the software development and cost analysis communities, and former program managers of major defense programs. The second method involved researching various resources to obtain specific program data to validate the problems identified by the professionals.

The data from the interviews came from three categories of professionals: cost analysis, program management, and academia. Of the six cost analysts interviewed, the experience estimating software cost ranged from two to twenty years. Five of the analysts are currently government personnel. Of those, two are assigned to program offices and the other three work in a Cost Analysis office. One of the cost analysts works for a prime contractor of a major defense program. The program managers interviewed included two officers who were former program managers of major defense programs, and two civilians who are currently working for major defense acquisition programs. The program managers' experience spans ten to twenty years with both ACAT 1, 2 and 3 type programs. The instructor interviewed has been teaching software-related courses for several years at the Defense Acquisition University. These individuals are listed in Appendix B.

The specific program data comes from two sources. The first data is from a 2002 memorandum concerning contractor performance of a major defense program. This memorandum provided current quantitative data, which is relevant to most complex, software intensive, weapon systems of today. The second data source is from a General Accounting Office (GAO) audit of the Federal Aviation Administration's (FAA) Standard Terminal Automation Replacement System, conducted in January 1998. Both of these sources were useful in validating the problems identified by the individuals interviewed. There were many books, guidebooks, journal articles, and industry briefs that non-

quantitatively validated these findings. For example, Table 6, from the Air Force Software Technology Support Center, produced the *Guidelines for Successful Acquisition and Management of Software-Intensive Systems* (GSAM) that included non-quantitative but useful data on the software risk areas associated with software developments. Another example of non-quantitative data is shown in Table 7.

The remainder of this Chapter is divided into software cost estimating risk categories that were derived from the interviews and other sources. The risk categories are organized first by listing problems identified by the individuals interviewed. Comments provided by cost analysts and academia will be referenced as "professionals," and program management individuals will be referenced as "management." Second, information in the program data paragraph will be referenced as "defense program" or "FAA program."

A. REQUIREMENTS

One of the leading causes of inaccurate software cost estimates identified by both groups interviewed, related to program requirements. The following paragraphs provide data on the various problems associated with requirements.

1. Interviews

Both professionals and management indicated that program requirements were usually poorly defined and highly unstable, making it extremely difficult to estimate the software development cost. The professionals further stated that DoD software intensive programs include advanced technologies that make the estimating process more complex. They stated that historical data normally used to compare programs and calibrate software cost estimating models were not readily available for advanced technologies. One professional said that a primary problem within Government program offices is that there is no requirements development framework.

Management stated that budget cuts, politics, and changing user requirements were constantly affecting requirements baseline. Management also stated that having too many bosses affected requirements. Another problem identified by management was that improper assumptions were made on requirements.

Program Level	**Excessive, immature, unrealistic, or unstable requirements** **Lack of user involvement** **Underestimation of program complexity or dynamic nature**
Program Attributes	**Performance shortfalls (includes defects and quality)** **Unrealistic cost or schedule (estimates and/or allocated amounts)**
Management	**Ineffective program management (multiple levels possible)**
Engineering	**Ineffective integration, assembly and test, quality control, specialty engineering, or systems engineering (multiple levels possible)** **Unanticipated difficulties associated with the user interface**
Work Environment	**Immature or untried design, process, or technologies selected** **Inadequate work plans or configuration control** **Inappropriate methods or tool selection or inaccurate metrics** **Poor training**
Other	**Inadequate or excessive documentation or review process** **Legal or contractual issues (such as litigation, malpractice, ownership) Obsolescence (includes excessive schedule length)** **Unanticipated difficulties with subcontracted items** **Unanticipated maintenance and/or support costs**

Table 6. Software Risk Areas [After: Ref. 9]

Lags in the adoption of fundamental metrics
Lags in the productivity measurement technology
Schedules are longer than any other kind of software project
Productivity is lower than for any other industry
Contracts for software have the highest rates of challenges and litigation
Contractors rank first in layoffs and downsizing
Contractors lag in staff benefits and compensation
Contractors lag in training and education of technical staff
Contractors lag in training of project managers
Contracted software has the highest growth of creeping user requirements
Contractors less effective at the Software Engineering Institute maturity levels compared to civilian performance-based contracts

Table 7. Factors Where DoD Software Lags Behind Commercial Programs [After: Ref.24]

2. Program Data

The following paragraphs include requirements-related data from the FAA and defense program that can negatively impact cost. The GAO audit stated that the FAA program had to increase their estimate for new and modified code by 50 percent after the first two years based on improper requirements assumptions. The GAO audit stated the FAA program requirements increased from the first day the development schedule was set. The audit also indicated the user was dissatisfied with the product. [Ref. 31]

With the defense program, the memorandum stated the contractor had management errors that included poor requirements definition prior to coding, and design problems that relied too heavily on a separate program being developed in parallel. The result was schedule and manpower estimates increased by 35 Percent.

B. SCHEDULE

Determining the success of a program is determined by the following three parameters: on-time, on-budget, and performs according to requirements. Having a realistic schedule is necessary in order to achieve success. The following paragraphs identify schedule risk that ultimately impact cost.

1. Interviews

Analyst and management indicated that exaggerated productivity rates proposed by contractors impacted schedules. The analyst also stated that historical data to validate productivity rates was not available. Management indicated that most contractor software delivery schedules are unrealistic, and that, most of the time, contractors proposed exaggerated productivity rates to win contracts.

2. Program Data

These paragraphs include data from the FAA and defense program that could adversely impact the schedule cost. The audit determined that the FAA program improperly developed their schedule by working backwards from a predetermined date rather than by estimating the schedule based on the size and complexity of the software development. Another problem with the FAA program was that productivity rates were 57 percent less than the projected 240 SLOC per man month.

With the defense program, the memorandum stated that schedules were primarily based on exaggerated productivity levels. For example, the contractor proposal indicated they could produce 300 SLOC per man-month, which included designing, coding, testing and integrating, but only realized 120 SLOC. The memorandum also stated the contractor relied heavily on reuse software code from another complex program that was still in development to further justify an optimistic schedule.

C. PROGRAM PLANNING

Initial program planning is essential to produce software that meets cost, schedule and performance goals. The following paragraphs outline risks associated with program planning that negatively impact cost.

1. Interviews

Management indicated that poor planning at the beginning of the program had lasting impacts over the life of the program. They stated the root cause of poor planning is inadequately trained staff. For example, management indicated that many of the software managers and cost analysts were not trained in the latest development processes and software estimating tools that help develop plans. Likewise, the professionals indicated that many of the program managers were not trained sufficiently to manage software intensive weapon systems.

2. Program Data

These paragraphs include data from the FAA and defense program that indicate poor planning. The audit reported the contractor for the FAA program initially proposed unrealistic productivity rates. The contractor probably considered this to be strategic planning instead of poor planning. The audit stated that it was poor planning for the contractor to wait three months to assign a software manager to the program. The contractor's timing for implementing a new corporate software development tool early in the life of the program was also questioned. The program eliminated Partial System Test 1 and compressed the schedule for other test. The air traffic controllers experienced low user satisfaction with the computer-human interface.

The memorandum for the defense program listed five examples of poor planning on the part of the contractor which include: 1) initially proposed unrealistic schedules and

productivity rates; 2) relied heavily on another program under development for software reuse; 3) systems engineering and software engineering activities were scattered; 4) inexperienced in software development methodologies for the programming language; and 5) overall staff was poorly trained.

D. SOFTWARE MAINTAINANCE AND SUPPORTABILITY

Software maintenance and support can account for over 70 percent of the total life cycle costs for a software system. [Ref. 22] The following paragraphs identify problems that impact cost associated with software maintenance and support.

1. Interviews

The professionals interviewed indicated that poorly defined programs at the beginning, along with unstable requirements, caused software estimates for maintenance and support to be inaccurate. They said that this occurred primarily because the software cost estimates were based on initial estimates of source lines of code or number of functions to be performed by the development.

2. Program Data

These paragraphs include data from the FAA and defense program that could affect maintenance and support cost in the future. The FAA program eliminated Partial System Test 1 and compressed the schedule for other tests, including Partial System Test 2, Installation and Integration Tests, and Site Acceptance Tests. The contractor introduced a new corporate software development tool early in the life of the program. The memorandum stated the defense program contractor initially had poor requirements definition prior to coding. The contractor's staff were poorly trained and inexperienced in the development methodology for the programming language required.

The defense program's contractor had a poorly trained staff and the programmers were inexperienced in the software methodology employed. The program experienced poor requirements definition prior to coding.

E. DATA SUMMARY

Chapter IV will analyze the cost implications by risk category for each of the problems identified by the professionals, and management personnel. Data from the

GAO audit of the FAA program, and the defense program, will also be analyzed. Chapter V provides recommendations and conclusions to avoid these problems.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ANALYSIS OF DATA

Chapter III outlined problems associated with software cost estimating inaccuracies identified by professionals in the software development and cost analysis communities, and former program managers of major defense programs. Chapter III also included actual data from two complex software intensive programs. This chapter will analyze each category of risk identified in Chapter III as well as the potential cost impacts to a program, for each of the problems identified.

A. REQUIREMENTS ANALYSIS

1. Requirements Definition

The first phase in most software life cycle development programs is requirements analysis and specification. When requirements analysis is performed poorly, it permeates throughout the life cycle of the program. For example, if the poorly defined requirements make it into the request for proposal (RFP), the contractor who was awarded the contract by the Government would almost certainly have a severely underestimated development program.

2. SLOC and Function Points Estimates

Most methods for estimating software costs use SLOC or function points as a parameter to calculate cost. Therefore, if the requirements are unstable or poorly defined, estimating SLOC and function points is difficult because these initial parameter estimates will drive the cost for the entire software development. For example, the FAA program hoped to use 85 percent COTS software and 15 percent new/modified code. However, by ultimately using less COTS and increasing new/modified code by 50 percent, the original cost estimate increased by 33 percent.

3. Advanced Technology Impact

Uncertainties associated with advanced technologies within a program make it difficult to conduct requirements analysis and design programs. Once again, this usually results in underestimated SLOC and function points. The problem is compounded when improper complexity factors are applied to the SLOC and function point estimates. The

risk for these estimates increase because there is often no historical data against which the estimate can be compared.

4. User Involvement

Management and data from the audit of the FAA program indicated that changing requirements from the user impacted requirements analysis. Obviously, the user was not adequately included in the requirements, design, development and test phases of the program. This resulted in a product being delivered that did not meet the users requirements. Having to modify code later in the development phase is more expensive.

5. Requirements Development Framework

One of the professionals mentioned that the Government programs do not always have a requirements development framework. The requirements framework will ensure that traceability of the requirements is controlled. The impact of not controlling requirements is the possibility that requirements may be added or omitted without properly documenting the change.

6. Budget Cuts and Politics

Management indicated that additional problems with managing requirements resulted when too many levels of supervision were involved in the management process. People external to the program office will often impose changes to the program. For example, budget cuts may reduce the capabilities of the program, which results in a failure to meet requirements. Additionally, politics may dictate where the program spends its money, causing critical requirements to be delayed.

7. Improper Assumptions

Improper initial assumptions in the requirements phase typically lead to cost and schedule problems. For example, the FAA program assumed that 85 percent of the development would be simple COTS. When the COTS estimate was decreased and the new/modified code increased two years later, the cost increased by 33 percent over the original estimate.

B. SCHEDULE REALISM

1. Unrealistic Schedules

The realism of the schedule is a key requirement to successfully manage software intensive program. The contractor must have the personnel, facilities, experience, and the time required to deliver the software product. Any limitation to the above factors will cause the schedule to slip, and increase cost for the program. For example, the data indicated that the defense program lacked experienced using the programming language methodology. As a consequence, the productivity rate originally proposed at 300 SLOC per man month decreased to less than 120 SLOC per man month. The contractor then had to re-baseline at the realized production rate, and the program office was forced to increase their schedule time by 35 percent.

2. Exaggerated Productivity Rates

Both contractors for the FAA and defense program exaggerated their productivity rates. One month, the FAA contractor productivity was 57 percent of the projected 240 SLOC per man month. Obviously, at this rate it will take nearly twice as long to complete the program. As a result, the cost increases because, although the contractor is producing less, they continue to charge the Government the same rate per hour.

3. Backing into Schedules

Backing into the schedule, or compressing the schedule is not a problem if the program and contractor have sufficient resources. For example, the program manager's schedule may be compressed by one year for any given reason. The program manager has adequate funding to provide the contractor, however, the contractor may not have the facilities or manpower to meet the new schedule. Therefore, even having unlimited funding will not always get the job done. [Ref. 26]

C. INITIAL PROGRAMMING PLANNING

1. Poor Planning and Processes

Poor planning and process will ultimately affect program costs because it will be difficult to estimate SLOC and function points. For example, if SLOC and function points are underestimated, then used to calculate maintenance and support cost, the total life cycle cost would be severely underestimated. The development will be costlier and the resulting code will be larger. Larger code typically requires more support, and life cycle cost suffers.

2. Staffing and Training Problems

Management indicated the primary reason for poorly planned programs was inadequately trained staff. Likewise, the professionals stated the program managers were not sufficiently trained to manage software intensive programs. Properly trained personnel are required to plan and manage the complex processes, and effectively and efficiently allocate resources. Poor planning during the requirements and design phase will have lasting cost implications throughout the life cycle.

The same holds true for the contractors. Both the FAA and defense program had staffing and training problems that resulted in cost and schedule overruns. The FAA contractor waited three months to assign a software manager, and had to spend time training the staff to operate a new software development tool. If these two activities had been better planned, the cost may have decreased.

3. Reuse and COTS

Planning to utilize reuse and COTS code is considered good planning. However, both management and professionals, along with data from both programs, indicated that the program and the contractor often underestimate the utility of using COTS and reuse. When this occurs, schedules and cost become unrealistic. The cost estimates increase substantially when new/modified code must be developed.

D. SOFTWARE MAINTENACNE AND SUPPORTABILITY

1. Initial Unstable Requirements

The professionals indicated that poorly defined initial requirements was the chief reason for underestimated software maintenance and support cost. This simply goes back to underestimating SLOC and function points. When those estimates are inaccurate, the entire estimate is inaccurate.

2. Initial Design

Software needs to be designed to be reliable, understandable, and modifiable. All three of these factors lower the cost of maintenance and support. For example, the defense program contactor had a poorly trained staff with inexperience in the programming language methodology. It follows that this would impact how well the code was written. Also, both programs underestimated SLOC based on COTS and reuse

designs, which caused the maintenance and support cost to be significantly underestimated at the end of the program.

3. Testing Requirements

Testing takes place throughout the development of the program. Some of these tests include unit, integration, system, and acceptance testing. If the contractor neglects to perform any of these tests, the result is typically software that lacks full functionality. The resources required to remedy the problems will severely impact the cost, schedule and performance of the program. The GAO audit stated that the typical outcome, in this event, is an increase in the expected development cost, but a decrease in quality, and a program that is behind schedule. Ref. 31]

E. DATA ANALYSIS SUMMARY

Chapter V will summarize Chapters III and IV, and present recommendations that should improve the accuracy of software cost estimates. Chapter V will also readdress the primary and secondary research questions.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

The primary research question for this thesis was centered on the problems that cause software cost estimates to be inaccurate and recommended solutions that are available to the program manager. The secondary questions were designed to focus on the critical cost drivers that have a significant impact on cost. The answers to these questions are located in Chapters III and IV, and are divided into four risk categories: 1) requirements; 2) schedule; 3) program planning; and 4) maintainability and supportability. This Chapter includes conclusions and recommendations for each of the primary risk categories, and an overall conclusion of the thesis.

A. REQUIREMENTS ANALYSIS STABILITY

Everyone interviewed for this thesis stated that the primary problem in estimating software was poorly defined and unstable requirements. There were numerous articles and industry briefs to support this premise. For example, the Standish Group reported in their analysis that the major reason that software projects fail is because the requirements were incomplete. [Ref. 2] When requirements are unstable and incomplete, it has been demonstrated that both Government and contractors will inaccurately estimate SLOC and function points. For most software cost estimating methods, those are the critical parameters required to calculate cost. For example, when the FAA program experienced problems, and the SLOC count increased by 50 percent, the original cost estimate increased by 33 percent. [Ref. 31]

The analysis also concluded that the following activities contributed to unstable and changing requirements: 1) user's mission continually evolving; 2) lack of user participation in program IPTs; 3) incorporation of new and advanced technologies; 4) changing budgets; 5) political influences; 6) and improper initial assumptions. Any and potentially all of these activities can significantly impact the program. The program manager should continuously assess what can go wrong with the program.

Even well-defined programs experience requirements growth. For example, analysis of several thousand applications during benchmark and baseline studies

determined that requirements creep averaged 2 percent per month. [Ref. 24] These small monthly changes can have significant cost, schedule and performance impacts throughout the development of the software. [Ref. 9]

Program managers should understand that changes in requirements are inevitable. Therefore, the program manager should design a software architecture that is flexible and tolerant to change and also develop a requirements framework plan. These documents should be written in a clear, concise, and quantifiable manner. These documents and processes will serve as a primer to ensure requirements are testable and compliant with mandatory standards. The documentation will also provide the basis to manage and measure the success of the program. There are many program management, software development, and software cost estimating tools available to assist the program manager. When choosing a commercial model the program manager should select one that meets the organization's needs. Typical commercial model capabilities are listed in Table 8.

B. SCHEDULE REALISM

The software cost estimate is based on the program following a given schedule. Any time there is a deviation in schedule there is a cost impact. The leading cause of schedule slips identified during the research was contractors exaggerating their software productivity rates. Typically, the contractor's productivity rate is measured by how many SLOC or function points can be written by one person in one month. When the contractor exaggerates this number, as they did in both of the programs discussed in this thesis, the impact is an increase in both costs and schedule. The contractor must have the people, experience, facilities, and time to deliver the software product.

The first recommendation for the program manager is to plan and document the entire program. This would include scheduling, resources, development activities, test activities, software configuration management, software quality management, risk management, training, deployment, support and maintenance. A complete baseline of measures should be performed at the beginning of the program, to include, level of effort, number of COTS components, personnel profiles, project characteristics, rating standard used, risk, software product size, and total software cost. Once these baselines are

established, effectiveness should be tracked and measured throughout the program. [Ref. 22]

Most Models include:	Some Models Also Include:
Sizing logic for specifications, source code, and test cases. Phase-level, activity-level, and task-level estimation. Support for both function point metrics and source lines-of-code (SLOC) metrics. Support for specialized metrics such as object-oriented metrics. Support for backfiring or conversion between SLOC and function points. Support for software reusability of various artifacts. Support for traditional languages such as COBOL and FORTRAN. Support for modern languages such as JAVA and Visual Basic. Quality and reliability estimation.	Risk and value analysis. Estimation templates derived from historical data. Links to project management tools such as Artemis or Microsoft Project. Cost and time-to-complete estimates mixing historical data with projected data. Currency conversions for international projects. Inflation calculations for long-term projects. Estimates keyed to the Software Engineering Institute's Capability Maturity Model® (CMM®)

Table 8. Commercial Model Estimating Capabilities [After: Ref. 24]

The second recommendation is to use commercial software development and cost estimating models to develop baseline estimates. For best results, it is recommended that an IPT be used to determine which values/factors should be input into the model. It was highly recommended by the professionals that two or more models be used to compare results.

Having this information prior to sending out a RFP will provide the program office a basis to compare proposals. It is also recommended that the Government define in the RFP what is considered a SLOC or function point. One method is to use a matrix similar to Table 2. [Ref. 9] Additionally, the RFP should include a matrix similar to

Table 9 to determine the programmer/developer years of experience for a given software language. [Ref. 27]

The proposals should include information about the contractor's CMM level. A contractor with a CMM level IV would most likely have a more realistic cost and schedule proposal. For a complex software intensive program, the program office should consider contractors with CMM levels of IV or V. A contractor certified at level III may propose a lower bid, but they most likely would not be the best value. For example, studies have shown that improving one CMM level can reduce software development cost from 4-11 percent. [Ref. 28] Contractors, achieving level 5 are positioned to maximize quality and productivity for developmental efforts. [Ref. 29] All of these tools will make it easier for the program office to compare proposals and select the contractor with the highest probability of delivering the product on-time, on-budget, and fully mission capable.

Experience	Ada 83			Ada 9X			OO Methodology			CASE Tools		
Years	<1	1-5	6-10+	<1	1-2	3+	<1	1-5	6-10+	<1	1-5	6-10+
Prime Contractor												
Subcontractor												
Total												

Table 9. Example of Developers Years Experience [From: Ref. 27]

Once proposals are received, one of the managers suggested using a benchmarking tool along with estimating model results to determine if the proposals are realistic. The manager stated the tool is also useful after contract award to monitor productivity levels. The database for the benchmarking tool in Table 10 is composed of 500 projects completed in the last seven years. The data is scoped to include requirements analysis, architectural design, development, and software integration and test. Note, all the data is presented in ranges. Donald Reifer stated in *Crosstalk Journal*, that although an average productivity of SLOC/staff month is provided, the benchmarking tool is best used to determine if a productivity level falls within a range for a given application. Reifer also expressed concern that people would misquote or use the numbers incorrectly. Therefore, it is strongly recommended that anyone who wishes to use this data, should refer to the March 2002 *Crosstalk Journal* to avoid incorrect interpretation of the numbers. [Ref. 30]

Application Domain	Number of Projects	Size Range (KSLOC)	Average Productivity (SLOC/SM)	Range (SLOC/SM)	Example Applications
Automation	55	25-650	245	120-440	Factory automation
Command & Control	43	35-4,500	225	95-330	Command centers
Data Processing	36	20-780	330	165-500	Business systems
Envrionment/Tools	75	15-1,200	260	143-610	CASE tool, compilers
Military-Airborne	38	20-1,350	105	65-250	Embedded sensors
Military-Ground	52	25-2,125	195	80-300	Combat information center
Military-Missile	14	22-125	85	52-165	Guidance, navigation
Military-Spaceborne	18	15-465	90	45-175	Attitude control system
Scientific	33	28-790	195	130-360	Seismic processing sys.
Telecomunications	48	15-1,800	250	175-440	Digital switches
Trainers/Simulations	24	200-900	224	143-780	Virtual reality sim.
Web	64	10-270	275	190-975	Client/server sites

Table 10. Software Productivity (SLOC/staff month) [After: Ref. 30]

C. PROGRAMMING AND PLANNING

Poor programming and planning effects software cost estimates in the same manner as requirements and schedule. They all affect the SLOC and function point estimates. One method to mitigate risk associated with programming and planning is to use the Sixteen Critical Software Practices ™ for performance-based management, developed by the Software Program Managers Network. The sixteen practices include: 1) adopt continuous program risk management; 2) estimate cost and schedule empirically; 3) use metrics to manage; 4) track earned value; 5) track defects against quality targets; 6) treat people as the most important resource; 7) adopt life cycle configuration management; 8) manage and trace requirements; 9) use system-based software design; 10) ensure data and database interoperability; 11) define and control interfaces; 12) design twice, code once; 13) assess reuse risks and costs; 14) inspect requirements and design; 15) manage testing as a continuous process; and 16) compile and smoke test frequently. [Ref. 22]

One of the professionals interviewed stated that Government program offices should implement the Software Acquisition (SA) CMM within their program. One ACAT 1 program recognized that imposing the CMM process on the contractor could not reap all the benefits without the program office also instituting the SA CMM. By focusing the project office on standardized process, the program manager was able to turn around a struggling software development and achieve a successful milestone B decision.

The SEI led a group of Government and industry leaders to develop, pilot-test, and plan the implementation of the SA-CMM. [Ref. 23]

All of the professionals and managers indicated that having an adequately trained staff is critical for a program to have a successful software development. In the past, program managers were inadequately trained to manage complex software intensive programs. However, the DoD recognized this deficiency and has been aggressively educating officers and civilians to mitigate this problem. Therefore, it is recommended that future program manager should have advanced degrees, preferably in program management.

Successful implementation of the various plans identified in Paragraph B requires the staff to be trained in those processes. There are many short courses available from the Defense Acquisition University (DAU) in the areas of program management, software acquisition and cost analysis. Several of these courses are available through distance learning over the Internet. [Ref. 32] Acquisition personnel are required to attend many of these courses to become certified level III in their primary field.

The program manager should ensure that all of the staffs training plans are current. Personnel should be encouraged to take advantage of DAU sponsored courses. For example, the software cost estimating class teaches acquisition personnel to develop and evaluate cost estimates for life cycle management, plan and manage DoD system acquisitions, evaluate and negotiate contract proposals, and analyze cost and performance tradeoffs.

Training is also required for the sophisticated commercial software estimating models. The program manager should ensure that cost and technical software personnel have a thorough understanding of how the model works. It is also imperative that personnel know how to calibrate the models to maximize their effectiveness.

One analyst interviewed reported that the program manager, of a major ACAT 1 program recognized that imposing the CMM process on the contractor could not reap all the benefits without the program office also instituting the Software Acquisition CMM. By focusing the project office on standardized processes, he was able to turn around a struggling software development, and achieve a successful milestone B decision.

Implementing any and all of these plans, processes, and procedures will improve the probability of success for the program.

D. SOFTWARE MAINTAINABLITY AND SUPPORTABILITY

Software maintenance and support is a significant cost driver over the total life of the program. It is imperative that software developers design software to be reliable, understandable and modifiable. Improving the initial design will hopefully reduce the trend where maintenance and support costs are 40-80 percent of the total software life cycle cost. Figure 12 represents current maintenance and support data for data processing programs. In both cases the maintenance costs for these programs would be significant.

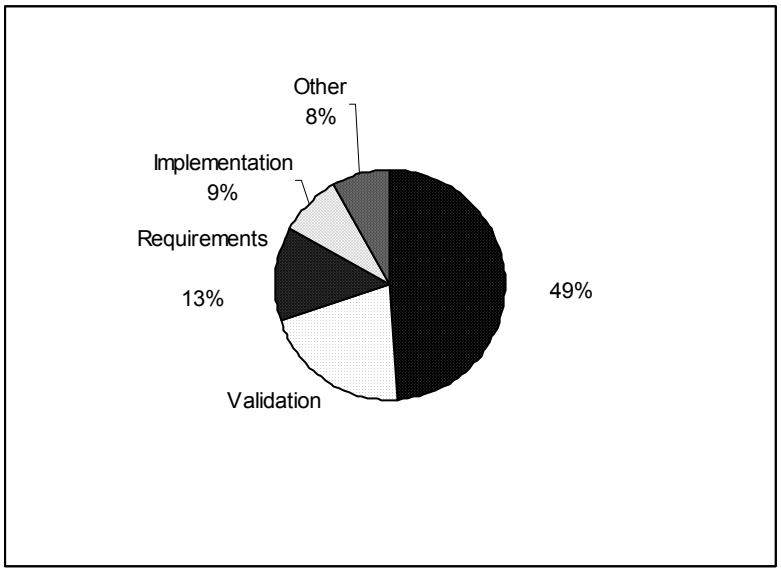


Figure 12. Support Cost for Data Processing Environments [After: Ref. 9]

It is recommended that modern software estimating tools be used to estimate the total life cycle cost of the program. However, when comparing results with other models or against other programs, it important to understand how the models differ on techniques to estimate total life cycle cost and in particularly what is included in the maintenance and support cost. A comprehensive list of support activities is provided in Table 11 to determine what level of support is required for the program. [Ref. From: 9]

1	Maintainability	Requirement for a Maintenance Task Analysis (MTA)
2	FTA, FMECA	Requirement for Fault Tree Analysis (FTA) and Failure Modes and Effects and Criticality Analysis (FMECA) to be performed to functional dept
3	Defect Rate	Requirements to state a contractual target defect rate per lines of code over an agreed period including confidence limits
4	Failure Identification	Design to provide features that achieve failure detection and location times
5	Failure Snapshot	Design to provide features that achieve failure detection and location times
6	Tool Kit	Provision of User/Maintainers software tool kits to aid failure location
7	Loading and Saving Data	Design to allow loading or saving data in specified times
8	Config. Identification	User/maintainer able to identify the configuration status (version) without accompanying documentation
9	Exception Handling	Design to allow exception handling to preclude failure conditions from aborting software during operations
10	Sup. Policy Constraint	Use Study to include what the software must do and not do
11	Support Maint. Policy	Support specific maintenance policies and manpower ceilings and skill level availability to be stated
12	SW Sup. & Maint. Cat.	Categories of software support and maintenance to be stated
13	Media	Proposed media must: (a) suit the environmental requirements, and (b) be acceptable as a consumable item
14	Media Copying	Simplify copying and distribution
15	Media Marking	To allow physical and internal marking; safety critical items to be separately marked
16	Packaging	Media packaging to be consumable, reusable, and robust
17	Handling	Media to require no special precautions and meet Use Study requirements
18	Storage	Media to require no special precautions or facilities and meet Use Study requirements
19	Transportation	Media and packaging to require no special requirements
20	Training, User	User training required to detect failures and invoke exception handling
21	Training, Support	Support training required to detect and locate failures and invoke exception handling
22	Publications	User and Support publications will be required
23	Definitions	The Requirement must include contractually agreed upon definitions of: incident, fault, failure, defect, reliability, and failure categories
24	Resources	Cost estimates must be sought for software maintenance
25	Test Tools	Contractor-owned and maintained software test tools and documentation must be provided
26	Test Tool Access	Access to test tools to be provided to software support personnel
27	Incident/Failure Reporting	Incident and failure reporting to be available

Table 11. Software Supportability Checklist [From: Ref. 9]

E. SUMMARY

Staffing, training, processes and tools are the keys to improving software development programs and improving the accuracy of the software cost estimates. For multi-million dollar programs, the program manager should purchase a couple of modern estimating tools, and hire a professional consulting firm to develop a customized benchmarking tool.

The program manager should establish a cost IPT that includes representatives from each of the technical areas of the program. The cost analyst should also participate in all of the technical IPTs. Because many program offices have small staffs, it may be necessary to award a contract for a Systems Engineering Technical Assistance (SETA) support contractor. With SETA support, the cost team can effectively attend all the meetings, run the cost models and collect data required to calibrate the models.

F. RECOMMENDATIONS FOR FURTHER ANALYSIS

Recommendations for further analysis include: 1) examining the implementation of the Software Acquisition CMM for Government program offices; 2) investigating maintenance and support cost to reduce cost; 3) and developing a case study on a program that is successfully implementing a suite of modern program management, software development and cost models.

G. VALAUBLE RESOUCES

Some of the more valuable resources that would benefit the program manager include; The Parametric Estimating Handbook; Joint Industry/Government, Spring 1999, The Guidelines for Successful Acquisition and Management of Software-Intensive Systems, (GSAM) Version 3.0, May 2000, (both of these are in the Defense Acquisition Deskbook CD, March 2002), The Program Manager's Guide for Managing Software, and the monthly *CrossTalk Journals*, and the Software Engineering Institute's website.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. INDIVIDUALS INTERVIEWED

Mr. Jason Wilson
Cost Analysis
Research Development Acquisition Office
Space and Missile Defense Command

Mr. Randy Mills
Cost Analysis
Research Development Acquisition Office
Space and Missile Defense Command

Ms. Robbie Holcomb
Cost Analysis
Research Development Acquisition Office
Mr. Ed Strange

Ms. Beverly Fuller
Operations Research Analyst
Program Executive Office, Tactical Missiles

Mr. Gary Fuller
Manager
Future Combat Systems
T&E Test Resources & Facilities
The Boeing Company

Mr. Ken Shipman
Software Program Manager
JLENS Project Office
Program Executive Office, Air and Missile Defense

Mr. Jerome Olerich
Software Program Manager
Comanche Project Office
Program Executive Office, Aviation

Ms. Martha Spurlock
Software Cost Estimating and Statistics Instructor
Defense Acquisition University
Mid Atlantic Region Fort Lee Center

LTC (ret) Brad Naegle
Graduate School of Business & Public Policy
Naval Postgraduate School

COL (ret) Dave Matthews
Graduate School of Business & Public Policy
Naval Postgraduate School

APPENDIX B. SOFTWARE COST ESTIMATING MODEL WEBSITES

Sage
http://www.seisage.com/sage.htm

CostXPert
http://www.costxpert.com/

COCOMO II
http://sunset.usc.edu/research/COCOMOII/

PRICE S
http://www.pricesystems.com/

SEER
http://www.galorath.com/home.shtm

REVIC and SoftEST
http://sepo.spawar.navy.mil/sepo/estimation.html

SLIM
http://www.qsm.com/

THIS PAGE INTENTIONALLY LEFT

LIST OF REFERENCES

1. Information Resources Management College, National Defense University, Slaying the Software Dragon, by Michel, John, Lt. Col., 1998

2. The Standish Group, T23E-T10E, Standish Group Report, 1995

3. Boehm, Barry, *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981

4. Department of Defense, Department of Defense Directive (DoDD) 5000.2-R Defense Acquisition, Washington, D.C., 1999

5. The Deputy Secretary of Defense Memorandum, Subject: Defense Acquisition, Oct. 2002

6. The Deputy Secretary of Defense, *Interim Defense Acquisition Guidebook*, Oct. 2002

7. Defense Contract Management Command, *Parametric Estimating Handbook Joint Industry/Government*, Spring 1991

8. Department of Defense Information Officer Memorandum, *Use of the Ada Programming Language*, April 1997

9. Department of the Air Force, Software Technology Support Center, *Guidelines for Successful Acquisition and Management of Software-Intensive Systems, (GSAM)* Version 3.0, May 2000

10. Pressman, Roger, *Software Engineering A Practitioner's Approach*, McGraw Hill, Boston, MA, 2001

11. Department of Defense, The Strategic Defense Initiative Organization, *Software Cost Estimating: Life Cycle, Models, and Techniques Training Manual*, April 1993

12. Boehm, Barry, and others *Software Cost Estimation With COCOMO II*, Prentice Hall, Upper Saddle River, NJ, 2000

13. Software Engineering Institute, CMU/SEI-92-TR-19, ESC-TR-92-017, *Software Measurement for DoD Systems: Recommended for Initial Core Measures*, Carleton, Anita, and others, 1992

14. Joint Single Integrated Air Picture, Systems Engineering Task Force, *Cost Benefit Analysis Questionnaire*, adapted from CostXpert Model, 2002

15. REVIC, Software Cost Estimating Model User's Manual, ver. 9, 1991

16. *SEER Quick Reference Guide*, Version 1.31, Galorath Associates, Inc., Los Angeles, CA, 1992

17. *Price S Estimating Cost and Schedule Guide,* 1st ed., Price Systems, L.L.C., Mt. Laurel, NJ, 1998

18. DoD 5000.4-M, *Guidelines for the Preparation and Maintenance of a Cost Analysis Requirements Document (CARD),* 1992

19. Dean, Joe, "Software Models What Model is Right for Me*?,"* brief presented at Software Engineering Institute Conference, 2000

20. Appleyard, James, Project Management Resources for the Program Manager, http://www.projectmagazine.com/sept01/costx1.html, 10 Dec. 2002

21. Ramgolam, Rakhee, *A Guide to Selecting Software Metrics for the Acqusisition of Weapon Systems*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 2001

22. Department of Defense, Program Manager's Guide for Managing Software, Draft 0.6, June 2001

23. Software Engineering Institute, *Software Acquisition Capability Maturity Model,*® http://www.sei.cmu.edu/arm/SA-CMM.html, 10 December, 2002

24. Jones, Capers, "Software Cost Estimation in 2002", Crosstalk, The Journal of Defense Software Engineering, Vol. 15 No.6, June 2002

25. Stark, George, and others, *An Examination of the Effects of Requirements Changes on Software Maintenance Releases, h*ttp://members.aol.com/GEShome/ ibelieve/jsmregres.PDF, 7 December, 2002

26. Puttman, Lawrence; and Myers, Ware, "Control the Software Beast With Metrics-Based Management", Crosstalk, The Journal of Defense Software Engineering, Vol. 15 No.8, August 2002

27. Department of Defense, Ada 95 Adoption Handbook; A Guide to Investigating Ada 95; Ver. 1.2, September, 1995

28. Clark, Brad, "Quantifying the Effects on Effort of Process Improvement." IEEE Software Nov./Dec. 2000

29. Diaz , Mark; King, Jeff, "How CMM Impacts Quality, Productivity, Rework, and the Bottom Line", Crosstalk, The Journal of Defense Software Engineering, Vol. 15 No.3, March 2002

30. Reifer, Donald, "Let the Numbers Do the Talking", Crosstalk, The Journal of Defense Software Engineering, Vol. 15 No.3, March 2002

31. General Accounting Office, Air Traffic Control: Timely Completion of FAA's Standard Terminal Automation Replacement System Software Is at Risk, GAO/AIMD-98-41R, January 1998

32. Defense Acquisition University (DAU) Website, Course Catalog, http://www.dau.mil/ catalog/ cat2003/Chapter4.pdf8, December 15, 2002

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Mr. Brad Naegle
   Naval Postgraduate School
   Monterey, CA

4. Mr. William C. Reeves
   U.S. Army Space and Missile Defense Command
   Huntsville, AL

5. Mr. Richard H. Brown
   U.S. Army Space and Missile Defense Command
   Huntsville, AL

6. Dr. Latika Becker
   U.S. Army Space and Missile Defense Command
   Huntsville, AL